

GL_Scene

Generated by Doxygen 1.13.2

1 Namespace Index	1
1.1 Namespace List	1
2 Hierarchical Index	3
2.1 Class Hierarchy	3
3 Class Index	5
3.1 Class List	5
4 File Index	7
4.1 File List	7
5 Namespace Documentation	9
5.1 half_float Namespace Reference	9
5.1.1 Detailed Description	11
5.1.2 Function Documentation	12
5.1.2.1 abs()	12
5.1.2.2 acos()	13
5.1.2.3 acosh()	13
5.1.2.4 asin()	14
5.1.2.5 asinh()	14
5.1.2.6 atan()	14
5.1.2.7 atan2()	15
5.1.2.8 atanh()	15
5.1.2.9 cbrt()	16
5.1.2.10 ceil()	16
5.1.2.11 copysign()	17
5.1.2.12 cos()	17
5.1.2.13 cosh()	17
5.1.2.14 erf()	18
5.1.2.15 erfc()	18
5.1.2.16 exp()	19
5.1.2.17 exp2()	19
5.1.2.18 expm1()	20
5.1.2.19 fabs()	20
5.1.2.20 fdim()	20
5.1.2.21 feclearexcept()	21
5.1.2.22 fegetexceptflag()	21
5.1.2.23 feraiseexcept()	22
5.1.2.24 fesetexceptflag()	22
5.1.2.25 fetestexcept()	22
5.1.2.26 fethrowexcept()	23
5.1.2.27 floor()	23
5.1.2.28 fma()	24

5.1.2.29 <code>fmax()</code>	24
5.1.2.30 <code>fmin()</code>	25
5.1.2.31 <code>fmod()</code>	25
5.1.2.32 <code>fpclassify()</code>	25
5.1.2.33 <code>frexp()</code>	26
5.1.2.34 <code>half_cast()</code> [1/2]	26
5.1.2.35 <code>half_cast()</code> [2/2]	27
5.1.2.36 <code>hypot()</code> [1/2]	28
5.1.2.37 <code>hypot()</code> [2/2]	28
5.1.2.38 <code>ilogb()</code>	29
5.1.2.39 <code>isfinite()</code>	29
5.1.2.40 <code>isgreater()</code>	29
5.1.2.41 <code>isgreaterequal()</code>	30
5.1.2.42 <code>isinf()</code>	30
5.1.2.43 <code>isless()</code>	30
5.1.2.44 <code>islessequal()</code>	31
5.1.2.45 <code>islessgreater()</code>	31
5.1.2.46 <code>isnan()</code>	31
5.1.2.47 <code>isnormal()</code>	32
5.1.2.48 <code>isunordered()</code>	32
5.1.2.49 <code>ldexp()</code>	32
5.1.2.50 <code>lgamma()</code>	33
5.1.2.51 <code>log()</code>	33
5.1.2.52 <code>log10()</code>	34
5.1.2.53 <code>log1p()</code>	34
5.1.2.54 <code>log2()</code>	35
5.1.2.55 <code>logb()</code>	35
5.1.2.56 <code>lrint()</code>	36
5.1.2.57 <code>lround()</code>	36
5.1.2.58 <code>modf()</code>	36
5.1.2.59 <code>nanh()</code>	37
5.1.2.60 <code>nearbyint()</code>	37
5.1.2.61 <code>nextafter()</code>	37
5.1.2.62 <code>nexttoward()</code>	38
5.1.2.63 <code>operator!=(())</code>	38
5.1.2.64 <code>operator*()</code>	39
5.1.2.65 <code>operator+()</code> [1/2]	39
5.1.2.66 <code>operator+()</code> [2/2]	40
5.1.2.67 <code>operator-()</code> [1/2]	40
5.1.2.68 <code>operator-()</code> [2/2]	40
5.1.2.69 <code>operator/()</code>	41
5.1.2.70 <code>operator<()</code>	41

5.1.2.71 operator<<()	42
5.1.2.72 operator<=()	42
5.1.2.73 operator==()	42
5.1.2.74 operator>()	43
5.1.2.75 operator>=()	43
5.1.2.76 operator>>()	44
5.1.2.77 pow()	44
5.1.2.78 remainder()	45
5.1.2.79 remquo()	45
5.1.2.80 rint()	46
5.1.2.81 round()	46
5.1.2.82 rsqrt()	46
5.1.2.83 scalbln()	47
5.1.2.84 scalbn()	47
5.1.2.85 signbit()	48
5.1.2.86 sin()	48
5.1.2.87 sincos()	49
5.1.2.88 sinh()	49
5.1.2.89 sqrt()	50
5.1.2.90 tan()	50
5.1.2.91 tanh()	50
5.1.2.92 tgamma()	51
5.1.2.93 trunc()	51
5.2 std Namespace Reference	52
5.2.1 Detailed Description	52
6 Class Documentation	53
6.1 half_float::detail::binary_t Struct Reference	53
6.1.1 Detailed Description	53
6.2 half_float::detail::bits< T > Struct Template Reference	53
6.2.1 Detailed Description	54
6.3 half_float::detail::bits< const T > Struct Template Reference	54
6.4 half_float::detail::bits< const volatile T > Struct Template Reference	54
6.5 half_float::detail::bits< double > Struct Reference	55
6.5.1 Detailed Description	55
6.6 half_float::detail::bits< float > Struct Reference	55
6.6.1 Detailed Description	56
6.7 half_float::detail::bits< volatile T > Struct Template Reference	56
6.8 half_float::detail::bool_type< bool > Struct Template Reference	56
6.8.1 Detailed Description	56
6.9 Camera Class Reference	57
6.9.1 Detailed Description	57

6.9.2 Constructor & Destructor Documentation	58
6.9.2.1 Camera()	58
6.9.3 Member Function Documentation	58
6.9.3.1 get_view_matrix()	58
6.9.3.2 process_keyboard()	58
6.9.3.3 process_mouse_movement()	59
6.9.4 Member Data Documentation	59
6.9.4.1 front	59
6.9.4.2 mouse_sensitivity	59
6.9.4.3 movement_speed	59
6.9.4.4 pitch	59
6.9.4.5 position	60
6.9.4.6 right	60
6.9.4.7 up	60
6.9.4.8 world_up	60
6.9.4.9 yaw	60
6.9.4.10 zoom	60
6.10 half_float::detail::conditional< bool, T, typename > Struct Template Reference	61
6.10.1 Detailed Description	61
6.11 half_float::detail::conditional< false, T, F > Struct Template Reference	61
6.12 Cube Class Reference	61
6.12.1 Detailed Description	62
6.13 udit::Cube Class Reference	62
6.13.1 Constructor & Destructor Documentation	63
6.13.1.1 Cube() [1/4]	63
6.13.1.2 Cube() [2/4]	64
6.13.1.3 Cube() [3/4]	64
6.13.1.4 Cube() [4/4]	64
6.14 EventHandler Class Reference	65
6.14.1 Detailed Description	65
6.14.2 Constructor & Destructor Documentation	65
6.14.2.1 EventHandler()	65
6.14.3 Member Function Documentation	65
6.14.3.1 handle_events()	65
6.15 half_float::detail::f31 Struct Reference	66
6.15.1 Detailed Description	66
6.15.2 Constructor & Destructor Documentation	66
6.15.2.1 f31() [1/2]	66
6.15.2.2 f31() [2/2]	67
6.15.3 Friends And Related Symbol Documentation	67
6.15.3.1 operator*	67
6.15.3.2 operator+	67

6.15.3.3 operator-	68
6.15.3.4 operator/	68
6.16 half_float::half Class Reference	68
6.16.1 Detailed Description	71
6.16.2 Constructor & Destructor Documentation	71
6.16.2.1 half() [1/2]	71
6.16.2.2 half() [2/2]	71
6.16.3 Member Function Documentation	71
6.16.3.1 operator float()	71
6.16.3.2 operator*=() [1/2]	72
6.16.3.3 operator*=() [2/2]	72
6.16.3.4 operator++() [1/2]	72
6.16.3.5 operator++() [2/2]	73
6.16.3.6 operator+=() [1/2]	73
6.16.3.7 operator+=() [2/2]	73
6.16.3.8 operator--() [1/2]	74
6.16.3.9 operator--() [2/2]	74
6.16.3.10 operator-=() [1/2]	74
6.16.3.11 operator-=() [2/2]	75
6.16.3.12 operator/=() [1/2]	75
6.16.3.13 operator/=() [2/2]	76
6.16.3.14 operator=()	76
6.16.4 Friends And Related Symbol Documentation	77
6.16.4.1 acos	77
6.16.4.2 acosh	77
6.16.4.3 asin	78
6.16.4.4 asinh	78
6.16.4.5 atan	78
6.16.4.6 atan2	79
6.16.4.7 atanh	79
6.16.4.8 cbrt	80
6.16.4.9 ceil	80
6.16.4.10 copysign	81
6.16.4.11 cos	81
6.16.4.12 cosh	81
6.16.4.13 erf	82
6.16.4.14 erfc	82
6.16.4.15 exp	83
6.16.4.16 exp2	83
6.16.4.17 expm1	84
6.16.4.18 fabs	84
6.16.4.19 fdim	84

6.16.4.20 floor	85
6.16.4.21 fma	85
6.16.4.22 fmax	86
6.16.4.23 fmin	86
6.16.4.24 fmod	87
6.16.4.25 fpclassify	87
6.16.4.26 frexp	88
6.16.4.27 hypot [1/2]	88
6.16.4.28 hypot [2/2]	89
6.16.4.29 ilogb	89
6.16.4.30 isfinite	90
6.16.4.31 isgreater	90
6.16.4.32 isgreaterequal	90
6.16.4.33 isinf	91
6.16.4.34 isless	91
6.16.4.35 islessequal	91
6.16.4.36 islessgreater	92
6.16.4.37 isnan	92
6.16.4.38 isnormal	92
6.16.4.39 lgamma	93
6.16.4.40 log	93
6.16.4.41 log10	94
6.16.4.42 log1p	94
6.16.4.43 log2	95
6.16.4.44 logb	95
6.16.4.45 lrint	96
6.16.4.46 lround	96
6.16.4.47 modf	96
6.16.4.48 nanh	97
6.16.4.49 nearbyint	97
6.16.4.50 nextafter	97
6.16.4.51 nexttoward	98
6.16.4.52 operator"!=	98
6.16.4.53 operator*	99
6.16.4.54 operator+	99
6.16.4.55 operator- [1/2]	100
6.16.4.56 operator- [2/2]	100
6.16.4.57 operator/	100
6.16.4.58 operator<	101
6.16.4.59 operator<<	101
6.16.4.60 operator<=	102
6.16.4.61 operator==	102

6.16.4.62 operator>	103
6.16.4.63 operator>=	103
6.16.4.64 operator>>	104
6.16.4.65 pow	104
6.16.4.66 remainder	105
6.16.4.67 remquo	105
6.16.4.68 rint	106
6.16.4.69 round	106
6.16.4.70 rsqrt	106
6.16.4.71 scalbln	107
6.16.4.72 signbit	107
6.16.4.73 sin	108
6.16.4.74 sincos	108
6.16.4.75 sinh	109
6.16.4.76 sqrt	109
6.16.4.77 tan	109
6.16.4.78 tanh	110
6.16.4.79 tgamma	110
6.16.4.80 trunc	111
6.17 half_float::detail::half_caster< T, U, R > Struct Template Reference	111
6.17.1 Detailed Description	111
6.18 half_float::detail::half_caster< half, half, R > Struct Template Reference	112
6.19 half_float::detail::half_caster< half, U, R > Struct Template Reference	112
6.20 half_float::detail::half_caster< T, half, R > Struct Template Reference	112
6.21 half_float::detail::is_float< typename > Struct Template Reference	113
6.21.1 Detailed Description	113
6.22 half_float::detail::is_float< const T > Struct Template Reference	113
6.23 half_float::detail::is_float< const volatile T > Struct Template Reference	114
6.24 half_float::detail::is_float< double > Struct Reference	114
6.25 half_float::detail::is_float< float > Struct Reference	114
6.26 half_float::detail::is_float< long double > Struct Reference	115
6.27 half_float::detail::is_float< volatile T > Struct Template Reference	115
6.28 udit::Light Class Reference	115
6.28.1 Detailed Description	116
6.28.2 Constructor & Destructor Documentation	116
6.28.2.1 Light()	116
6.28.3 Member Function Documentation	116
6.28.3.1 make_light()	116
6.28.3.2 send_to_shader()	117
6.29 Mesh Class Reference	117
6.29.1 Detailed Description	119
6.29.2 Constructor & Destructor Documentation	119

6.29.2.1 Mesh()	119
6.29.2.2 ~Mesh()	119
6.29.3 Member Function Documentation	119
6.29.3.1 create_mesh()	119
6.29.3.2 get_model_view_matrix()	120
6.29.3.3 get_shader_matrix_ids()	120
6.29.3.4 get_shader_program_id()	120
6.29.3.5 make_mesh()	120
6.29.3.6 render()	120
6.29.3.7 resize()	121
6.29.3.8 rotate()	121
6.29.3.9 scale()	121
6.29.3.10 set_mesh_type()	121
6.29.3.11 set_model_view_matrix()	122
6.29.3.12 set_shader()	122
6.29.3.13 translate()	122
6.29.3.14 update()	123
6.30 udit::Mesh Class Reference	123
6.30.1 Detailed Description	124
6.30.2 Constructor & Destructor Documentation	124
6.30.2.1 Mesh()	124
6.30.2.2 ~Mesh()	125
6.30.3 Member Function Documentation	125
6.30.3.1 create_mesh()	125
6.30.3.2 get_model_view_matrix()	125
6.30.3.3 get_shader_matrix_ids()	125
6.30.3.4 get_shader_program_id()	126
6.30.3.5 make_mesh()	126
6.30.3.6 render()	126
6.30.3.7 resize()	126
6.30.3.8 rotate()	127
6.30.3.9 scale()	127
6.30.3.10 set_mesh_type()	127
6.30.3.11 set_model_view_matrix()	127
6.30.3.12 set_shader()	128
6.30.3.13 translate()	128
6.30.3.14 update()	128
6.31 std::numeric_limits< half_float::half > Class Reference	128
6.31.1 Detailed Description	130
6.32 udit::Window::OpenGL_Context_Settings Struct Reference	130
6.33 Window::OpenGL_Context_Settings Struct Reference	131
6.34 Plane Class Reference	131

6.34.1 Detailed Description	133
6.34.2 Constructor & Destructor Documentation	133
6.34.2.1 Plane() [1/3]	133
6.34.2.2 Plane() [2/3]	133
6.34.2.3 Plane() [3/3]	133
6.35 udit::Plane Class Reference	134
6.35.1 Detailed Description	136
6.35.2 Constructor & Destructor Documentation	136
6.35.2.1 Plane() [1/3]	136
6.35.2.2 Plane() [2/3]	136
6.35.2.3 Plane() [3/3]	136
6.36 Scene Class Reference	137
6.36.1 Detailed Description	137
6.36.2 Constructor & Destructor Documentation	137
6.36.2.1 Scene()	137
6.36.3 Member Function Documentation	138
6.36.3.1 render()	138
6.36.3.2 resize()	138
6.36.3.3 set_lights()	138
6.36.3.4 set_projection_matrix()	138
6.36.3.5 set_view_matrix()	139
6.36.3.6 update()	139
6.37 udit::Scene Class Reference	139
6.37.1 Detailed Description	140
6.37.2 Constructor & Destructor Documentation	140
6.37.2.1 Scene()	140
6.37.3 Member Function Documentation	140
6.37.3.1 render()	140
6.37.3.2 resize()	140
6.37.3.3 set_lights()	141
6.37.3.4 set_projection_matrix()	141
6.37.3.5 set_view_matrix()	141
6.37.3.6 update()	142
6.38 Shader Class Reference	142
6.38.1 Detailed Description	143
6.38.2 Constructor & Destructor Documentation	143
6.38.2.1 Shader() [1/2]	143
6.38.2.2 Shader() [2/2]	143
6.38.2.3 ~Shader()	143
6.38.3 Member Function Documentation	144
6.38.3.1 compile_shaders()	144
6.38.3.2 get_model_view_matrix_id()	144

6.38.3.3	get_name()	144
6.38.3.4	get_normal_matrix_id()	144
6.38.3.5	get_program_id()	145
6.38.3.6	get_projection_matrix_id()	145
6.38.3.7	has_textures()	145
6.38.3.8	make_shader()	145
6.38.3.9	set_name()	146
6.38.3.10	set_texture()	147
6.38.3.11	set_texture_scale()	147
6.38.3.12	use()	147
6.39	udit::Shader Class Reference	147
6.39.1	Detailed Description	148
6.39.2	Constructor & Destructor Documentation	149
6.39.2.1	Shader() [1/2]	149
6.39.2.2	Shader() [2/2]	149
6.39.2.3	~Shader()	149
6.39.3	Member Function Documentation	149
6.39.3.1	compile_shaders()	149
6.39.3.2	get_model_view_matrix_id()	150
6.39.3.3	get_name()	150
6.39.3.4	get_normal_matrix_id()	150
6.39.3.5	get_program_id()	150
6.39.3.6	get_projection_matrix_id()	151
6.39.3.7	has_textures()	151
6.39.3.8	make_shader()	151
6.39.3.9	set_name()	151
6.39.3.10	set_texture()	152
6.39.3.11	set_texture_scale()	152
6.39.3.12	use()	152
6.40	Skybox Class Reference	152
6.40.1	Detailed Description	154
6.40.2	Constructor & Destructor Documentation	154
6.40.2.1	Skybox() [1/2]	154
6.40.2.2	Skybox() [2/2]	154
6.40.3	Member Function Documentation	155
6.40.3.1	getCubemapTexture()	155
6.41	udit::Skybox Class Reference	155
6.41.1	Detailed Description	157
6.41.2	Constructor & Destructor Documentation	157
6.41.2.1	Skybox() [1/2]	157
6.41.2.2	Skybox() [2/2]	157
6.41.3	Member Function Documentation	158

6.41.3.1 getCubemapTexture()	158
6.42 Texture Class Reference	158
6.42.1 Detailed Description	159
6.42.2 Constructor & Destructor Documentation	159
6.42.2.1 Texture()	159
6.42.3 Member Function Documentation	159
6.42.3.1 bind()	159
6.42.3.2 is_loaded()	159
6.42.3.3 load_texture()	160
6.42.3.4 set_type()	160
6.42.3.5 unbind()	160
6.43 udit::Texture Class Reference	160
6.43.1 Detailed Description	161
6.43.2 Constructor & Destructor Documentation	161
6.43.2.1 Texture()	161
6.43.3 Member Function Documentation	161
6.43.3.1 bind()	161
6.43.3.2 is_loaded()	162
6.43.3.3 load_texture()	162
6.43.3.4 set_type()	162
6.43.3.5 unbind()	162
6.44 udit::Window Class Reference	162
6.44.1 Constructor & Destructor Documentation	163
6.44.1.1 Window()	163
6.45 Window Class Reference	164
6.45.1 Constructor & Destructor Documentation	164
6.45.1.1 Window()	164
7 File Documentation	165
7.1 Camera.hpp	165
7.2 Cube.hpp	166
7.3 EventHandler.hpp	166
7.4 GL_Scene/half.hpp File Reference	166
7.4.1 Detailed Description	173
7.4.2 Macro Definition Documentation	173
7.4.2.1 FP_FAST_FMAH	173
7.4.2.2 HALF_ENABLE_F16C_INTRINSICS	173
7.4.2.3 HALF_ERRHANDLING_OVERFLOW_TO_INEXACT	173
7.4.2.4 HALF_ERRHANDLING_UNDERFLOW_TO_INEXACT	173
7.4.2.5 HALF_ROUND_STYLE	173
7.4.2.6 HALF_ROUNDING	174
7.4.2.7 HUGE_VALH	174

7.4.3 Function Documentation	174
7.4.3.1 angle_arg()	174
7.4.3.2 area()	174
7.4.3.3 arithmetic_shift()	175
7.4.3.4 atan2()	175
7.4.3.5 atan2_args()	176
7.4.3.6 builtin_isinf()	176
7.4.3.7 builtin_isnan()	176
7.4.3.8 builtin_signbit()	177
7.4.3.9 check_underflow()	177
7.4.3.10 compsignal()	178
7.4.3.11 divide64()	178
7.4.3.12 erf()	178
7.4.3.13 errflags()	179
7.4.3.14 exp2()	179
7.4.3.15 exp2_post()	180
7.4.3.16 fixed2half()	181
7.4.3.17 float2half()	182
7.4.3.18 float2half_impl() [1/3]	182
7.4.3.19 float2half_impl() [2/3]	183
7.4.3.20 float2half_impl() [3/3]	183
7.4.3.21 gamma()	184
7.4.3.22 half2float()	185
7.4.3.23 half2float_impl() [1/3]	186
7.4.3.24 half2float_impl() [2/3]	186
7.4.3.25 half2float_impl() [3/3]	186
7.4.3.26 half2int()	187
7.4.3.27 hyperbolic_args()	187
7.4.3.28 hypot_post()	188
7.4.3.29 int2half()	188
7.4.3.30 integral()	189
7.4.3.31 invalid()	189
7.4.3.32 log2()	190
7.4.3.33 log2_post()	190
7.4.3.34 mod()	191
7.4.3.35 mulhi()	191
7.4.3.36 multiply64()	192
7.4.3.37 overflow()	192
7.4.3.38 pole()	192
7.4.3.39 raise()	193
7.4.3.40 rounded()	193
7.4.3.41 select()	194

7.4.3.42 sign_mask()	194
7.4.3.43 signal() [1/3]	194
7.4.3.44 signal() [2/3]	195
7.4.3.45 signal() [3/3]	195
7.4.3.46 sincos()	196
7.4.3.47 sqrt()	196
7.4.3.48 tangent_post()	196
7.4.3.49 underflow()	197
7.5 half.hpp	198
7.6 Light.hpp	242
7.7 Mesh.hpp	243
7.8 Plane.hpp	244
7.9 GL_Scene/Scene.hpp File Reference	244
7.9.1 Detailed Description	245
7.10 Scene.hpp	245
7.11 GL_Scene/Shader.hpp File Reference	246
7.11.1 Detailed Description	246
7.11.2 Enumeration Type Documentation	246
7.11.2.1 ShaderType	246
7.12 Shader.hpp	247
7.13 GL_Scene/Skybox.hpp File Reference	248
7.13.1 Detailed Description	248
7.14 Skybox.hpp	248
7.15 GL_Scene/Texture.hpp File Reference	249
7.15.1 Detailed Description	249
7.15.2 Enumeration Type Documentation	249
7.15.2.1 Texture_Type	249
7.16 Texture.hpp	250
7.17 Window.hpp	250
Index	253

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

half_float	9
std	Extensions to the C++ standard library	52

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

half_float::detail::binary_t	53
half_float::detail::bits< T >	53
half_float::detail::bits< const T >	54
half_float::detail::bits< const volatile T >	54
half_float::detail::bits< volatile T >	56
half_float::detail::bits< double >	55
half_float::detail::bool_type< bool >	56
half_float::detail::bool_type< false >	56
half_float::detail::is_float< T >	113
half_float::detail::is_float< const T >	113
half_float::detail::is_float< const volatile T >	114
half_float::detail::is_float< volatile T >	115
half_float::detail::is_float< const T >	113
half_float::detail::is_float< const volatile T >	114
half_float::detail::is_float< double >	114
half_float::detail::is_float< float >	114
half_float::detail::is_float< long double >	115
half_float::detail::is_float< volatile T >	115
half_float::detail::is_float< typename >	113
half_float::detail::bool_type< true >	56
half_float::detail::is_float< double >	114
half_float::detail::is_float< float >	114
half_float::detail::is_float< long double >	115
Camera	57
half_float::detail::conditional< bool, T, typename >	61
half_float::detail::conditional< false, T, F >	61
half_float::detail::conditional< std::numeric_limits< unsigned int >::digits >	61
half_float::detail::bits< float >	55
Cube	61
EventHandler	65
half_float::detail::f31	66
half_float::half	68
half_float::detail::half_caster< T, U, R >	111
half_float::detail::half_caster< half, half, R >	112

half_float::detail::half_caster< half, U, R >	112
half_float::detail::half_caster< T, half, R >	112
udit::Light	115
Mesh	117
udit::Cube	62
udit::Skybox	155
udit::Plane	134
udit::Mesh	123
udit::Cube	62
udit::Plane	134
std::numeric_limits< half_float::half >	128
udit::Window::OpenGL_Context_Settings	130
Window::OpenGL_Context_Settings	131
Scene	137
udit::Scene	139
Shader	142
udit::Shader	147
Texture	158
udit::Texture	160
unsigned int	
half_float::detail::bits< float >	55
unsigned long	
half_float::detail::bits< float >	55
udit::Window	162
Window	164

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

half_float::detail::binary_t	
Tag type for binary construction	53
half_float::detail::bits< T >	
Type traits for floating-point bits	53
half_float::detail::bits< const T >	54
half_float::detail::bits< const volatile T >	54
half_float::detail::bits< double >	
Unsigned integer of (at least) 64 bits width	55
half_float::detail::bits< float >	
Unsigned integer of (at least) 32 bits width	55
half_float::detail::bits< volatile T >	56
half_float::detail::bool_type< bool >	
Helper for tag dispatching	56
Camera	
Clase que gestiona la cámara en un entorno 3D, incluyendo el control de la posición, orientación y el movimiento de la cámara	57
half_float::detail::conditional< bool, T, typename >	
Conditional type	61
half_float::detail::conditional< false, T, F >	61
Cube	
Clase que representa un cubo, heredando de la clase Mesh	61
udit::Cube	62
EventHandler	
Clase que maneja los eventos de entrada (teclado, ratón) en la escena	65
half_float::detail::f31	
Class for 1.31 unsigned floating-point computation	66
half_float::half	68
half_float::detail::half_caster< T, U, R >	111
half_float::detail::half_caster< half, half, R >	112
half_float::detail::half_caster< half, U, R >	112
half_float::detail::half_caster< T, half, R >	112
half_float::detail::is_float< typename >	
Type traits for floating-point types	113
half_float::detail::is_float< const T >	113
half_float::detail::is_float< const volatile T >	114

half_float::detail::is_float< double >	114
half_float::detail::is_float< float >	114
half_float::detail::is_float< long double >	115
half_float::detail::is_float< volatile T >	115
udit::Light	
Clase que representa una fuente de luz en la escena	115
Mesh	
Clase que representa una malla 3D	117
udit::Mesh	
Clase que representa una malla 3D	123
std::numeric_limits< half_float::half >	128
udit::Window::OpenGL_Context_Settings	130
Window::OpenGL_Context_Settings	131
Plane	
Clase que representa un plano 3D	131
udit::Plane	
Clase que representa un plano 3D	134
Scene	
Representa una escena 3D con un skybox, terreno, luz y otros elementos	137
udit::Scene	
Representa una escena 3D con un skybox, terreno, luz y otros elementos	139
Shader	
Representa un shader program en OpenGL	142
udit::Shader	
Representa un shader program en OpenGL	147
Skybox	
Representa un skybox, un cubo con texturas aplicadas en sus seis caras	152
udit::Skybox	
Representa un skybox, un cubo con texturas aplicadas en sus seis caras	155
Texture	
Representa una textura en OpenGL	158
udit::Texture	
Representa una textura en OpenGL	160
udit::Window	162
Window	164

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

GL_Scene/ Camera.hpp	165
GL_Scene/ Cube.hpp	166
GL_Scene/ EventHandler.hpp	166
GL_Scene/ half.hpp	166
GL_Scene/ Light.hpp	242
GL_Scene/ Mesh.hpp	243
GL_Scene/ Plane.hpp	244
GL_Scene/ Scene.hpp Clase que representa una escena 3D, gestionando objetos como el fondo, terreno, luz, etc . . .	244
GL_Scene/ Shader.hpp Clase que representa un shader en OpenGL, gestionando la compilación y uso de programas de sombreado	246
GL_Scene/ Skybox.hpp Clase para representar y gestionar un skybox en OpenGL	248
GL_Scene/ Texture.hpp Clase para gestionar las texturas en OpenGL	249
GL_Scene/ Window.hpp	250

Chapter 5

Namespace Documentation

5.1 half_float Namespace Reference

Classes

- class [half](#)

Functions

Comparison operators

- HALF_CONSTEXPR_NOERR bool [operator==](#) ([half](#) x, [half](#) y)
- HALF_CONSTEXPR_NOERR bool [operator!=](#) ([half](#) x, [half](#) y)
- HALF_CONSTEXPR_NOERR bool [operator<](#) ([half](#) x, [half](#) y)
- HALF_CONSTEXPR_NOERR bool [operator>](#) ([half](#) x, [half](#) y)
- HALF_CONSTEXPR_NOERR bool [operator<=](#) ([half](#) x, [half](#) y)
- HALF_CONSTEXPR_NOERR bool [operator>=](#) ([half](#) x, [half](#) y)

Arithmetic operators

- HALF_CONSTEXPR [half operator+](#) ([half](#) arg)
- HALF_CONSTEXPR [half operator-](#) ([half](#) arg)
- [half operator+](#) ([half](#) x, [half](#) y)
- [half operator-](#) ([half](#) x, [half](#) y)
- [half operator*](#) ([half](#) x, [half](#) y)
- [half operator/](#) ([half](#) x, [half](#) y)

Input and output

- template<typename charT, typename traits>
std::basic_ostream< charT, traits > & [operator<<](#) (std::basic_ostream< charT, traits > &out, [half](#) arg)
- template<typename charT, typename traits>
std::basic_istream< charT, traits > & [operator>>](#) (std::basic_istream< charT, traits > &in, [half](#) &arg)

Basic mathematical operations

- HALF_CONSTEXPR [half fabs](#) ([half](#) arg)
- HALF_CONSTEXPR [half abs](#) ([half](#) arg)
- [half fmod](#) ([half](#) x, [half](#) y)

- [half remainder](#) ([half](#) x, [half](#) y)
- [half remquo](#) ([half](#) x, [half](#) y, int *quo)
- [half fma](#) ([half](#) x, [half](#) y, [half](#) z)
- HALF_CONSTEXPR_NOERR [half fmax](#) ([half](#) x, [half](#) y)
- HALF_CONSTEXPR_NOERR [half fmin](#) ([half](#) x, [half](#) y)
- [half fdim](#) ([half](#) x, [half](#) y)
- [half nanh](#) (const char *arg)

Exponential functions

- [half exp](#) ([half](#) arg)
- [half exp2](#) ([half](#) arg)
- [half expm1](#) ([half](#) arg)
- [half log](#) ([half](#) arg)
- [half log10](#) ([half](#) arg)
- [half log2](#) ([half](#) arg)
- [half log1p](#) ([half](#) arg)

Power functions

- [half sqrt](#) ([half](#) arg)
- [half rsqrt](#) ([half](#) arg)
- [half cbrt](#) ([half](#) arg)
- [half hypot](#) ([half](#) x, [half](#) y)
- [half hypot](#) ([half](#) x, [half](#) y, [half](#) z)
- [half pow](#) ([half](#) x, [half](#) y)

Trigonometric functions

- void [sincos](#) ([half](#) arg, [half](#) *sin, [half](#) *cos)
- [half sin](#) ([half](#) arg)
- [half cos](#) ([half](#) arg)
- [half tan](#) ([half](#) arg)
- [half asin](#) ([half](#) arg)
- [half acos](#) ([half](#) arg)
- [half atan](#) ([half](#) arg)
- [half atan2](#) ([half](#) y, [half](#) x)

Hyperbolic functions

- [half sinh](#) ([half](#) arg)
- [half cosh](#) ([half](#) arg)
- [half tanh](#) ([half](#) arg)
- [half asinh](#) ([half](#) arg)
- [half acosh](#) ([half](#) arg)
- [half atanh](#) ([half](#) arg)

Error and gamma functions

- [half erf](#) ([half](#) arg)
- [half erfc](#) ([half](#) arg)
- [half lgamma](#) ([half](#) arg)
- [half tgamma](#) ([half](#) arg)

Rounding

- [half ceil](#) ([half](#) arg)
- [half floor](#) ([half](#) arg)
- [half trunc](#) ([half](#) arg)

- [half round](#) ([half](#) arg)
- long [lround](#) ([half](#) arg)
- [half rint](#) ([half](#) arg)
- long [lrint](#) ([half](#) arg)
- [half nearbyint](#) ([half](#) arg)

Floating point manipulation

- [half frexp](#) ([half](#) arg, int *[exp](#))
- [half scalbln](#) ([half](#) arg, long [exp](#))
- [half scalbn](#) ([half](#) arg, int [exp](#))
- [half ldexp](#) ([half](#) arg, int [exp](#))
- [half modf](#) ([half](#) arg, [half](#) *iptr)
- int [ilogb](#) ([half](#) arg)
- [half logb](#) ([half](#) arg)
- [half nextafter](#) ([half](#) from, [half](#) to)
- [half nexttoward](#) ([half](#) from, long double to)
- HALF_CONSTEXPR [half copysign](#) ([half](#) x, [half](#) y)

Floating point classification

- HALF_CONSTEXPR int [fpclassify](#) ([half](#) arg)
- HALF_CONSTEXPR bool [isfinite](#) ([half](#) arg)
- HALF_CONSTEXPR bool [isinf](#) ([half](#) arg)
- HALF_CONSTEXPR bool [isnan](#) ([half](#) arg)
- HALF_CONSTEXPR bool [isnormal](#) ([half](#) arg)
- HALF_CONSTEXPR bool [signbit](#) ([half](#) arg)

Comparison

- HALF_CONSTEXPR bool [isgreater](#) ([half](#) x, [half](#) y)
- HALF_CONSTEXPR bool [isgreaterequal](#) ([half](#) x, [half](#) y)
- HALF_CONSTEXPR bool [isless](#) ([half](#) x, [half](#) y)
- HALF_CONSTEXPR bool [islessequal](#) ([half](#) x, [half](#) y)
- HALF_CONSTEXPR bool [islessgreater](#) ([half](#) x, [half](#) y)
- HALF_CONSTEXPR bool [isunordered](#) ([half](#) x, [half](#) y)

Casting

- template<typename T, typename U>
T [half_cast](#) (U arg)
- template<typename T, std::float_round_style R, typename U>
T [half_cast](#) (U arg)

Error handling

- int [feclearexcept](#) (int excepts)
- int [fetestexcept](#) (int excepts)
- int [feraiseexcept](#) (int excepts)
- int [fegetexceptflag](#) (int *flagp, int excepts)
- int [fesetexceptflag](#) (const int *flagp, int excepts)
- void [fethrowexcept](#) (int excepts, const char *msg="")

5.1.1 Detailed Description

Main namespace for half-precision functionality. This namespace contains all the functionality provided by the library.

5.1.2 Function Documentation

5.1.2.1 `abs()`

```
HALF_CONSTEXPR half half_float::abs (  
    half arg) [inline]
```

Absolute value. **See also:** Documentation for [std::abs](#).

Parameters

<i>arg</i>	operand
------------	---------

Returns

absolute value of *arg*

5.1.2.2 acos()

```
half half_float::acos (  
    half arg) [inline]
```

Arc cosine function. This function is exact to rounding for all rounding modes.

See also: Documentation for [std::acos](#).

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

arc cosine value of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN or if $\text{abs}(arg) > 1$
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.3 acosh()

```
half half_float::acosh (  
    half arg) [inline]
```

Hyperbolic area cosine. This function is exact to rounding for all rounding modes.

See also: Documentation for [std::acosh](#).

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

area cosine value of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN or arguments < 1
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.4 asin()

```
half half_float::asin (
    half arg) [inline]
```

Arc sine. This function is exact to rounding for all rounding modes.

See also: Documentation for [std::asin](#).

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

arc sine value of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN or if $\text{abs}(arg) > 1$
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.5 asinh()

```
half half_float::asinh (
    half arg) [inline]
```

Hyperbolic area sine. This function is exact to rounding for all rounding modes.

See also: Documentation for [std::asinh](#).

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

area sine value of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.6 atan()

```
half half_float::atan (
    half arg) [inline]
```

Arc tangent function. This function is exact to rounding for all rounding modes.

See also: Documentation for [std::atan](#).

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

arc tangent value of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.7 atan2()

```
half half_float::atan2 (
    half y,
    half x) [inline]
```

Arc tangent function. This function may be 1 ULP off the correctly rounded exact result in $\sim 0.005\%$ of inputs for `std::round_to_nearest`, in $\sim 0.1\%$ of inputs for `std::round_toward_zero` and in $\sim 0.02\%$ of inputs for any other rounding mode.

See also: Documentation for `std::atan2`.

Parameters

<i>y</i>	numerator
<i>x</i>	denominator

Returns

arc tangent value

Exceptions

<i>FE_INVALID</i>	if <i>x</i> or <i>y</i> is signaling NaN
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.8 atanh()

```
half half_float::atanh (
    half arg) [inline]
```

Hyperbolic area tangent. This function is exact to rounding for all rounding modes.

See also: Documentation for `std::atanh`.

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

area tangent value of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN or if $\text{abs}(arg) > 1$
<i>FE_DIVBYZERO</i>	for +/-1
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.9 `cbrt()`

```
half half_float::cbrt (
    half arg) [inline]
```

Cubic root. This function is exact to rounding for all rounding modes.

See also: Documentation for `std::cbrt`.

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

cubic root of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_INEXACT</i>	according to rounding

5.1.2.10 `ceil()`

```
half half_float::ceil (
    half arg) [inline]
```

Nearest integer not less than half value. **See also:** Documentation for `std::ceil`.

Parameters

<i>arg</i>	half to round
------------	---------------

Returns

nearest integer not less than *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_INEXACT</i>	if value had to be rounded

5.1.2.11 copysign()

```
HALF_CONSTEXPR half half_float::copysign (
    half x,
    half y) [inline]
```

Take sign. **See also:** Documentation for [std::copysign](#).

Parameters

<i>x</i>	value to change sign for
<i>y</i>	value to take sign from

Returns

value equal to *x* in magnitude and to *y* in sign

5.1.2.12 cos()

```
half half_float::cos (
    half arg) [inline]
```

Cosine function. This function is exact to rounding for all rounding modes.

See also: Documentation for [std::cos](#).

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

cosine value of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN or infinity
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.13 cosh()

```
half half_float::cosh (
    half arg) [inline]
```

Hyperbolic cosine. This function is exact to rounding for all rounding modes.

See also: Documentation for [std::cosh](#).

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

hyperbolic cosine value of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.14 `erf()`

```
half half_float::erf (
    half arg) [inline]
```

Error function. This function may be 1 ULP off the correctly rounded exact result for any rounding mode in <0.5% of inputs.

See also: Documentation for `std::erf`.

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

error function value of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.15 `erfc()`

```
half half_float::erfc (
    half arg) [inline]
```

Complementary error function. This function may be 1 ULP off the correctly rounded exact result for any rounding mode in <0.5% of inputs.

See also: Documentation for `std::erfc`.

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

1 minus error function value of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.16 exp()

```
half half_float::exp (
    half arg) [inline]
```

Exponential function. This function is exact to rounding for all rounding modes.

See also: Documentation for [std::exp](#).

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

e raised to *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.17 exp2()

```
half half_float::exp2 (
    half arg) [inline]
```

Binary exponential. This function is exact to rounding for all rounding modes.

See also: Documentation for [std::exp2](#).

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

2 raised to *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.18 `expm1()`

```
half half_float::expm1 (
    half arg) [inline]
```

Exponential minus one. This function may be 1 ULP off the correctly rounded exact result in <0.05% of inputs for `std::round_to_nearest` and in <1% of inputs for any other rounding mode.

See also: Documentation for `std::expm1`.

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

e raised to *arg* and subtracted by 1

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.19 `fabs()`

```
HALF_CONSTEXPR half half_float::fabs (
    half arg) [inline]
```

Absolute value. **See also:** Documentation for `std::fabs`.

Parameters

<i>arg</i>	operand
------------	---------

Returns

absolute value of *arg*

5.1.2.20 `fdim()`

```
half half_float::fdim (
    half x,
    half y) [inline]
```

Positive difference. This function is exact to rounding for all rounding modes.

See also: Documentation for `std::fdim`.

Parameters

<i>x</i>	first operand
<i>y</i>	second operand

Returns

$x - y$ or 0 if difference negative

Exceptions

FE_{\leftarrow} _...	according to operator-(half,half)
---------------------------	---

5.1.2.21 `feclearexcept()`

```
int half_float::feclearexcept (
    int excepts) [inline]
```

Clear exception flags. This function works even if automatic exception flag handling is disabled, but in that case manual flag management is the only way to raise flags.

See also: Documentation for [std::feclearexcept](#).

Parameters

<i>excepts</i>	OR of exceptions to clear
----------------	---------------------------

Return values

0	all selected flags cleared successfully
---	---

5.1.2.22 `fegetexceptflag()`

```
int half_float::fegetexceptflag (
    int * flagp,
    int excepts) [inline]
```

Save exception flags. This function works even if automatic exception flag handling is disabled, but in that case manual flag management is the only way to raise flags.

See also: Documentation for [std::fegetexceptflag](#).

Parameters

<i>flagp</i>	adress to store flag state at
<i>excepts</i>	OR of flags to save

Return values

0	for success
---	-------------

5.1.2.23 `feraiseexcept()`

```
int half_float::feraiseexcept (
    int excepts) [inline]
```

Raise exception flags. This raises the specified floating point exceptions and also invokes any additional automatic exception handling as configured with the `HALF_ERRHANDLING_...` preprocessor symbols. This function works even if automatic exception flag handling is disabled, but in that case manual flag management is the only way to raise flags.

See also: Documentation for [std::feraiseexcept](#).

Parameters

<i>excepts</i>	OR of exceptions to raise
----------------	---------------------------

Return values

0	all selected exceptions raised successfully
---	---

5.1.2.24 `fesetexceptflag()`

```
int half_float::fesetexceptflag (
    const int * flagp,
    int excepts) [inline]
```

Restore exception flags. This only copies the specified exception state (including unset flags) without incurring any additional exception handling. This function works even if automatic exception flag handling is disabled, but in that case manual flag management is the only way to raise flags.

See also: Documentation for [std::fesetexceptflag](#).

Parameters

<i>flagp</i>	adress to take flag state from
<i>excepts</i>	OR of flags to restore

Return values

0	for success
---	-------------

5.1.2.25 `fetestexcept()`

```
int half_float::fetestexcept (
    int excepts) [inline]
```

Test exception flags. This function works even if automatic exception flag handling is disabled, but in that case manual flag management is the only way to raise flags.

See also: Documentation for [std::fetestexcept](#).

Parameters

<i>excepts</i>	OR of exceptions to test
----------------	--------------------------

Returns

OR of selected exceptions if raised

5.1.2.26 fethrowexcept()

```
void half_float::fethrowexcept (
    int excepts,
    const char * msg = "") [inline]
```

Throw C++ exceptions based on set exception flags. This function manually throws a corresponding C++ exception if one of the specified flags is set, no matter if automatic throwing (via `HALF_ERRHANDLING_THROW_...`) is enabled or not. This function works even if automatic exception flag handling is disabled, but in that case manual flag management is the only way to raise flags.

Parameters

<i>excepts</i>	OR of exceptions to test
<i>msg</i>	error message to use for exception description

Exceptions

<i>std::domain_error</i>	if <code>FE_INVALID</code> or <code>FE_DIVBYZERO</code> is selected and set
<i>std::overflow_error</i>	if <code>FE_OVERFLOW</code> is selected and set
<i>std::underflow_error</i>	if <code>FE_UNDERFLOW</code> is selected and set
<i>std::range_error</i>	if <code>FE_INEXACT</code> is selected and set

5.1.2.27 floor()

```
half half_float::floor (
    half arg) [inline]
```

Nearest integer not greater than half value. **See also:** Documentation for `std::floor`.

Parameters

<i>arg</i>	half to round
------------	---------------

Returns

nearest integer not greater than *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_INEXACT</i>	if value had to be rounded

5.1.2.28 fma()

```
half half_float::fma (
    half x,
    half y,
    half z) [inline]
```

Fused multiply add. This function is exact to rounding for all rounding modes.

See also: Documentation for `std::fma`.

Parameters

<i>x</i>	first operand
<i>y</i>	second operand
<i>z</i>	third operand

Returns

$(x * y) + z$ rounded as one operation.

Exceptions

<i>FE_INVALID</i>	according to <code>operator*()</code> and <code>operator+()</code> unless any argument is a quiet NaN and no argument is a signaling NaN
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding the final addition

5.1.2.29 fmax()

```
HALF_CONSTEXPR_NOERR half half_float::fmax (
    half x,
    half y) [inline]
```

Maximum of half expressions. **See also:** Documentation for `std::fmax`.

Parameters

<i>x</i>	first operand
<i>y</i>	second operand

Returns

maximum of operands, ignoring quiet NaNs

Exceptions

<i>FE_INVALID</i>	if <i>x</i> or <i>y</i> is signaling NaN
-------------------	--

5.1.2.30 fmin()

```
HALF_CONSTEXPR_NOERR half half_float::fmin (
    half x,
    half y) [inline]
```

Minimum of half expressions. **See also:** Documentation for [std::fmin](#).

Parameters

<i>x</i>	first operand
<i>y</i>	second operand

Returns

minimum of operands, ignoring quiet NaNs

Exceptions

<i>FE_INVALID</i>	if <i>x</i> or <i>y</i> is signaling NaN
-------------------	--

5.1.2.31 fmod()

```
half half_float::fmod (
    half x,
    half y) [inline]
```

Remainder of division. **See also:** Documentation for [std::fmod](#).

Parameters

<i>x</i>	first operand
<i>y</i>	second operand

Returns

remainder of floating-point division.

Exceptions

<i>FE_INVALID</i>	if <i>x</i> is infinite or <i>y</i> is 0 or if <i>x</i> or <i>y</i> is signaling NaN
-------------------	--

5.1.2.32 fpclassify()

```
HALF_CONSTEXPR int half_float::fpclassify (
    half arg) [inline]
```

Classify floating-point value. **See also:** Documentation for [std::fpclassify](#).

Parameters

<i>arg</i>	number to classify
------------	--------------------

Return values

<i>FP_ZERO</i>	for positive and negative zero
<i>FP_SUBNORMAL</i>	for subnormal numbers
<i>FP_INFINITY</i>	for positive and negative infinity
<i>FP_NAN</i>	for NaNs
<i>FP_NORMAL</i>	for all other (normal) values

5.1.2.33 frexp()

```
half half_float::frexp (
    half arg,
    int * exp) [inline]
```

Decompress floating-point number. **See also:** Documentation for `std::frexp`.

Parameters

<i>arg</i>	number to decompress
<i>exp</i>	address to store exponent at

Returns

significant in range [0.5, 1)

Exceptions

<i>FE_INVALID</i>	for signaling NaN
-------------------	-------------------

5.1.2.34 half_cast() [1/2]

```
template<typename T, typename U>
T half_float::half_cast (
    U arg)
```

Cast to or from half-precision floating-point number. This casts between `half` and any built-in arithmetic type. The values are converted directly using the default rounding mode, without any roundtrip over `float` that a `static_cast` would otherwise do.

Using this cast with neither of the two types being a `half` or with any of the two types not being a built-in arithmetic type (apart from `half`, of course) results in a compiler error and casting between `half`s returns the argument unmodified.

Template Parameters

<i>T</i>	destination type (half or built-in arithmetic type)
<i>U</i>	source type (half or built-in arithmetic type)

Parameters

<i>arg</i>	value to cast
------------	---------------

Returns

arg converted to destination type

Exceptions

<i>FE_INVALID</i>	if <i>T</i> is integer type and result is not representable as <i>T</i>
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.35 half_cast() [2/2]

```
template<typename T, std::float_round_style R, typename U>
T half_float::half_cast (
    U arg)
```

Cast to or from half-precision floating-point number. This casts between [half](#) and any built-in arithmetic type. The values are converted directly using the specified rounding mode, without any roundtrip over `float` that a `static_cast` would otherwise do.

Using this cast with neither of the two types being a [half](#) or with any of the two types not being a built-in arithmetic type (apart from [half](#), of course) results in a compiler error and casting between [halves](#) returns the argument unmodified.

Template Parameters

<i>T</i>	destination type (half or built-in arithmetic type)
<i>R</i>	rounding mode to use.
<i>U</i>	source type (half or built-in arithmetic type)

Parameters

<i>arg</i>	value to cast
------------	---------------

Returns

arg converted to destination type

Exceptions

<i>FE_INVALID</i>	if <i>T</i> is integer type and result is not representable as <i>T</i>
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.36 `hypot()` [1/2]

```
half half_float::hypot (
    half x,
    half y) [inline]
```

Hypotenuse function. This function is exact to rounding for all rounding modes.

See also: Documentation for `std::hypot`.

Parameters

<i>x</i>	first argument
<i>y</i>	second argument

Returns

square root of sum of squares without internal over- or underflows

Exceptions

<i>FE_INVALID</i>	if <i>x</i> or <i>y</i> is signaling NaN
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding of the final square root

5.1.2.37 `hypot()` [2/2]

```
half half_float::hypot (
    half x,
    half y,
    half z) [inline]
```

Hypotenuse function. This function is exact to rounding for all rounding modes.

See also: Documentation for `std::hypot`.

Parameters

<i>x</i>	first argument
<i>y</i>	second argument
<i>z</i>	third argument

Returns

square root of sum of squares without internal over- or underflows

Exceptions

<i>FE_INVALID</i>	if x, y or z is signaling NaN
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding of the final square root

5.1.2.38 ilogb()

```
int half_float::ilogb (
    half arg) [inline]
```

Extract exponent. **See also:** Documentation for [std::ilogb](#).

Parameters

<i>arg</i>	number to query
------------	-----------------

Returns

floating-point exponent

Return values

<i>FP_ILOGB0</i>	for zero
<i>FP_ILOGBNAN</i>	for NaN
<i>INT_MAX</i>	for infinity

Exceptions

<i>FE_INVALID</i>	for 0 or infinite values
-------------------	--------------------------

5.1.2.39 isfinite()

```
HALF_CONSTEXPR bool half_float::isfinite (
    half arg) [inline]
```

Check if finite number. **See also:** Documentation for [std::isfinite](#).

Parameters

<i>arg</i>	number to check
------------	-----------------

Return values

<i>true</i>	if neither infinity nor NaN
<i>false</i>	else

5.1.2.40 isgreater()

```
HALF_CONSTEXPR bool half_float::isgreater (
    half x,
    half y) [inline]
```

Quiet comparison for greater than. **See also:** Documentation for [std::isgreater](#).

Parameters

<i>x</i>	first operand
<i>y</i>	second operand

Return values

<i>true</i>	if <i>x</i> greater than <i>y</i>
<i>false</i>	else

5.1.2.41 isgreaterequal()

```
HALF_CONSTEXPR bool half_float::isgreaterequal (
    half x,
    half y) [inline]
```

Quiet comparison for greater equal. **See also:** Documentation for [std::isgreaterequal](#).

Parameters

<i>x</i>	first operand
<i>y</i>	second operand

Return values

<i>true</i>	if <i>x</i> greater equal <i>y</i>
<i>false</i>	else

5.1.2.42 isinf()

```
HALF_CONSTEXPR bool half_float::isinf (
    half arg) [inline]
```

Check for infinity. **See also:** Documentation for [std::isinf](#).

Parameters

<i>arg</i>	number to check
------------	-----------------

Return values

<i>true</i>	for positive or negative infinity
<i>false</i>	else

5.1.2.43 isless()

```
HALF_CONSTEXPR bool half_float::isless (
    half x,
    half y) [inline]
```

Quiet comparison for less than. **See also:** Documentation for [std::isless](#).

Parameters

<i>x</i>	first operand
<i>y</i>	second operand

Return values

<i>true</i>	if <i>x</i> less than <i>y</i>
<i>false</i>	else

5.1.2.44 islessequal()

```
HALF_CONSTEXPR bool half_float::islessequal (
    half x,
    half y) [inline]
```

Quiet comparison for less equal. **See also:** Documentation for [std::islessequal](#).

Parameters

<i>x</i>	first operand
<i>y</i>	second operand

Return values

<i>true</i>	if <i>x</i> less equal <i>y</i>
<i>false</i>	else

5.1.2.45 islessgreater()

```
HALF_CONSTEXPR bool half_float::islessgreater (
    half x,
    half y) [inline]
```

Quiet comparison for less or greater. **See also:** Documentation for [std::islessgreater](#).

Parameters

<i>x</i>	first operand
<i>y</i>	second operand

Return values

<i>true</i>	if either less or greater
<i>false</i>	else

5.1.2.46 isnan()

```
HALF_CONSTEXPR bool half_float::isnan (
    half arg) [inline]
```

Check for NaN. **See also:** Documentation for [std::isnan](#).

Parameters

<i>arg</i>	number to check
------------	-----------------

Return values

<i>true</i>	for NaNs
<i>false</i>	else

5.1.2.47 isnormal()

```
HALF_CONSTEXPR bool half_float::isnormal (
    half arg) [inline]
```

Check if normal number. **See also:** Documentation for [std::isnormal](#).

Parameters

<i>arg</i>	number to check
------------	-----------------

Return values

<i>true</i>	if normal number
<i>false</i>	if either subnormal, zero, infinity or NaN

5.1.2.48 isunordered()

```
HALF_CONSTEXPR bool half_float::isunordered (
    half x,
    half y) [inline]
```

Quiet check if unordered. **See also:** Documentation for [std::isunordered](#).

Parameters

<i>x</i>	first operand
<i>y</i>	second operand

Return values

<i>true</i>	if unordered (one or two NaN operands)
<i>false</i>	else

5.1.2.49 ldexp()

```
half half_float::ldexp (
    half arg,
    int exp) [inline]
```

Multiply by power of two. This function is exact to rounding for all rounding modes.

See also: Documentation for [std::ldexp](#).

Parameters

<i>arg</i>	number to modify
<i>exp</i>	power of two to multiply with

Returns

arg multiplied by 2 raised to *exp*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.50 lgamma()

```
half half_float::lgamma (
    half arg) [inline]
```

Natural logarithm of gamma function. This function may be 1 ULP off the correctly rounded exact result for any rounding mode in $\sim 0.025\%$ of inputs.

See also: Documentation for `std::lgamma`.

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

natural logarithm of gamma function for *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_DIVBYZERO</i>	for 0 or negative integer arguments
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.51 log()

```
half half_float::log (
    half arg) [inline]
```

Natural logarithm. This function is exact to rounding for all rounding modes.

See also: Documentation for `std::log`.

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

logarithm of *arg* to base e

Exceptions

<i>FE_INVALID</i>	for signaling NaN or negative argument
<i>FE_DIVBYZERO</i>	for 0
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.52 `log10()`

```
half half_float::log10 (
    half arg) [inline]
```

Common logarithm. This function is exact to rounding for all rounding modes.

See also: Documentation for `std::log10`.

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

logarithm of *arg* to base 10

Exceptions

<i>FE_INVALID</i>	for signaling NaN or negative argument
<i>FE_DIVBYZERO</i>	for 0
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.53 `log1p()`

```
half half_float::log1p (
    half arg) [inline]
```

Natural logarithm plus one. This function may be 1 ULP off the correctly rounded exact result in <0.05% of inputs for `std::round_to_nearest` and in ~1% of inputs for any other rounding mode.

See also: Documentation for `std::log1p`.

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

logarithm of *arg* plus 1 to base e

Exceptions

<i>FE_INVALID</i>	for signaling NaN or argument <-1
<i>FE_DIVBYZERO</i>	for -1
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.54 log2()

```
half half_float::log2 (  
    half arg) [inline]
```

Binary logarithm. This function is exact to rounding for all rounding modes.

See also: Documentation for [std::log2](#).

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

logarithm of *arg* to base 2

Exceptions

<i>FE_INVALID</i>	for signaling NaN or negative argument
<i>FE_DIVBYZERO</i>	for 0
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.55 logb()

```
half half_float::logb (  
    half arg) [inline]
```

Extract exponent. **See also:** Documentation for [std::logb](#).

Parameters

<i>arg</i>	number to query
------------	-----------------

Returns

floating-point exponent

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_DIVBYZERO</i>	for 0

5.1.2.56 `lrint()`

```
long half_float::lrint (
    half arg) [inline]
```

Nearest integer using half's internal rounding mode. **See also:** Documentation for `std::lrint`.

Parameters

<i>arg</i>	half expression to round
------------	--------------------------

Returns

nearest integer using default rounding mode

Exceptions

<i>FE_INVALID</i>	if value is not representable as `long`
<i>FE_INEXACT</i>	if value had to be rounded

5.1.2.57 `lround()`

```
long half_float::lround (
    half arg) [inline]
```

Nearest integer. **See also:** Documentation for `std::lround`.

Parameters

<i>arg</i>	half to round
------------	---------------

Returns

nearest integer, rounded away from zero in half-way cases

Exceptions

<i>FE_INVALID</i>	if value is not representable as `long`
-------------------	---

5.1.2.58 `modf()`

```
half half_float::modf (
    half arg,
    half * iptr) [inline]
```

Extract integer and fractional parts. **See also:** Documentation for `std::modf`.

Parameters

<i>arg</i>	number to decompress
<i>iptr</i>	address to store integer part at

Returns

fractional part

Exceptions

<i>FE_INVALID</i>	for signaling NaN
-------------------	-------------------

5.1.2.59 nanh()

```
half half_float::nanh (  
    const char * arg) [inline]
```

Get NaN value. **See also:** Documentation for `std::nan`.

Parameters

<i>arg</i>	string code
------------	-------------

Returns

quiet NaN

5.1.2.60 nearbyint()

```
half half_float::nearbyint (  
    half arg) [inline]
```

Nearest integer using half's internal rounding mode. **See also:** Documentation for `std::nearbyint`.

Parameters

<i>arg</i>	half expression to round
------------	--------------------------

Returns

nearest integer using default rounding mode

Exceptions

<i>FE_INVALID</i>	for signaling NaN
-------------------	-------------------

5.1.2.61 nextafter()

```
half half_float::nextafter (  
    half from,  
    half to) [inline]
```

Next representable value. **See also:** Documentation for `std::nextafter`.

Parameters

<i>from</i>	value to compute next representable value for
<i>to</i>	direction towards which to compute next value

Returns

next representable value after *from* in direction towards *to*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_OVERFLOW</i>	for infinite result from finite argument
<i>FE_UNDERFLOW</i>	for subnormal result

5.1.2.62 nexttoward()

```
half half_float::nexttoward (
    half from,
    long double to) [inline]
```

Next representable value. **See also:** Documentation for [std::nexttoward](#).

Parameters

<i>from</i>	value to compute next representable value for
<i>to</i>	direction towards which to compute next value

Returns

next representable value after *from* in direction towards *to*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_OVERFLOW</i>	for infinite result from finite argument
<i>FE_UNDERFLOW</i>	for subnormal result

5.1.2.63 operator!=(=)

```
HALF_CONSTEXPR_NOERR bool half_float::operator!=(= (
    half x,
    half y) [inline]
```

Comparison for inequality.

Parameters

<i>x</i>	first operand
<i>y</i>	second operand

Return values

<i>true</i>	if operands not equal
<i>false</i>	else

Exceptions

<i>FE_INVALID</i>	if <i>x</i> or <i>y</i> is NaN
-------------------	--------------------------------

5.1.2.64 operator*()

```
half half_float::operator* (
    half x,
    half y) [inline]
```

Multiplication. This operation is exact to rounding for all rounding modes.

Parameters

<i>x</i>	left operand
<i>y</i>	right operand

Returns

product of half expressions

Exceptions

<i>FE_INVALID</i>	if multiplying 0 with infinity or if <i>x</i> or <i>y</i> is signaling NaN
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.65 operator+() [1/2]

```
HALF_CONSTEXPR half half_float::operator+ (
    half arg) [inline]
```

Identity.

Parameters

<i>arg</i>	operand
------------	---------

Returns

unchanged operand

5.1.2.66 operator+() [2/2]

```
half half_float::operator+ (
    half x,
    half y) [inline]
```

Addition. This operation is exact to rounding for all rounding modes.

Parameters

<i>x</i>	left operand
<i>y</i>	right operand

Returns

sum of half expressions

Exceptions

<i>FE_INVALID</i>	if <i>x</i> and <i>y</i> are infinities with different signs or signaling NaNs
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.67 operator-() [1/2]

```
HALF_CONSTEXPR half half_float::operator- (
    half arg) [inline]
```

Negation.

Parameters

<i>arg</i>	operand
------------	---------

Returns

negated operand

5.1.2.68 operator-() [2/2]

```
half half_float::operator- (
    half x,
    half y) [inline]
```

Subtraction. This operation is exact to rounding for all rounding modes.

Parameters

<i>x</i>	left operand
<i>y</i>	right operand

Returns

difference of half expressions

Exceptions

<i>FE_INVALID</i>	if <i>x</i> and <i>y</i> are infinities with equal signs or signaling NaNs
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.69 operator/()

```
half half_float::operator/ (
    half x,
    half y) [inline]
```

Division. This operation is exact to rounding for all rounding modes.

Parameters

<i>x</i>	left operand
<i>y</i>	right operand

Returns

quotient of half expressions

Exceptions

<i>FE_INVALID</i>	if dividing 0s or infinities with each other or if <i>x</i> or <i>y</i> is signaling NaN
<i>FE_DIVBYZERO</i>	if dividing finite value by 0
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.70 operator<()

```
HALF_CONSTEXPR_NOERR bool half_float::operator< (
    half x,
    half y) [inline]
```

Comparison for less than.

Parameters

<i>x</i>	first operand
<i>y</i>	second operand

Return values

<i>true</i>	if <i>x</i> less than <i>y</i>
<i>false</i>	else

Exceptions

<i>FE_INVALID</i>	if <i>x</i> or <i>y</i> is NaN
-------------------	--------------------------------

5.1.2.71 operator<<()

```
template<typename charT, typename traits>
std::basic_ostream< charT, traits > & half_float::operator<< (
    std::basic_ostream< charT, traits > & out,
    half arg)
```

Output operator. This uses the built-in functionality for streaming out floating-point numbers.

Parameters

<i>out</i>	output stream to write into
<i>arg</i>	half expression to write

Returns

reference to output stream

5.1.2.72 operator<=()

```
HALF_CONSTEXPR_NOERR bool half_float::operator<= (
    half x,
    half y) [inline]
```

Comparison for less equal.

Parameters

<i>x</i>	first operand
<i>y</i>	second operand

Return values

<i>true</i>	if <i>x</i> less equal <i>y</i>
<i>false</i>	else

Exceptions

<i>FE_INVALID</i>	if <i>x</i> or <i>y</i> is NaN
-------------------	--------------------------------

5.1.2.73 operator==()

```
HALF_CONSTEXPR_NOERR bool half_float::operator==(
    half x,
    half y) [inline]
```

Comparison for equality.

Parameters

<i>x</i>	first operand
<i>y</i>	second operand

Return values

<i>true</i>	if operands equal
<i>false</i>	else

Exceptions

<i>FE_INVALID</i>	if <i>x</i> or <i>y</i> is NaN
-------------------	--------------------------------

5.1.2.74 operator>()

```
HALF_CONSTEXPR_NOERR bool half_float::operator> (
    half x,
    half y) [inline]
```

Comparison for greater than.

Parameters

<i>x</i>	first operand
<i>y</i>	second operand

Return values

<i>true</i>	if <i>x</i> greater than <i>y</i>
<i>false</i>	else

Exceptions

<i>FE_INVALID</i>	if <i>x</i> or <i>y</i> is NaN
-------------------	--------------------------------

5.1.2.75 operator>=()

```
HALF_CONSTEXPR_NOERR bool half_float::operator>= (
    half x,
    half y) [inline]
```

Comparison for greater equal.

Parameters

<i>x</i>	first operand
<i>y</i>	second operand

Return values

<i>true</i>	if <i>x</i> greater equal <i>y</i>
<i>false</i>	else

Exceptions

<i>FE_INVALID</i>	if <i>x</i> or <i>y</i> is NaN
-------------------	--------------------------------

5.1.2.76 `operator>>()`

```
template<typename charT, typename traits>
std::basic_istream< charT, traits > & half_float::operator>> (
    std::basic_istream< charT, traits > & in,
    half & arg)
```

Input operator. This uses the built-in functionality for streaming in floating-point numbers, specifically double precision floating point numbers (unless overridden with `HALF_ARITHMETIC_TYPE`). So the input string is first rounded to double precision using the underlying platform's current floating-point rounding mode before being rounded to half-precision using the library's half-precision rounding mode.

Parameters

<i>in</i>	input stream to read from
<i>arg</i>	half to read into

Returns

reference to input stream

Exceptions

<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding
--	-----------------------

5.1.2.77 `pow()`

```
half half_float::pow (
    half x,
    half y) [inline]
```

Power function. This function may be 1 ULP off the correctly rounded exact result for any rounding mode in $\sim 0.00025\%$ of inputs.

See also: Documentation for `std::pow`.

Parameters

<i>x</i>	base
<i>y</i>	exponent

Returns

x raised to *y*

Exceptions

<i>FE_INVALID</i>	if <i>x</i> or <i>y</i> is signaling NaN or if <i>x</i> is finite an negative and <i>y</i> is finite and not integral
<i>FE_DIVBYZERO</i>	if <i>x</i> is 0 and <i>y</i> is negative
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.78 remainder()

```
half half_float::remainder (
    half x,
    half y) [inline]
```

Remainder of division. **See also:** Documentation for `std::remainder`.

Parameters

<i>x</i>	first operand
<i>y</i>	second operand

Returns

remainder of floating-point division.

Exceptions

<i>FE_INVALID</i>	if <i>x</i> is infinite or <i>y</i> is 0 or if <i>x</i> or <i>y</i> is signaling NaN
-------------------	--

5.1.2.79 remquo()

```
half half_float::remquo (
    half x,
    half y,
    int * quo) [inline]
```

Remainder of division. **See also:** Documentation for `std::remquo`.

Parameters

<i>x</i>	first operand
<i>y</i>	second operand
<i>quo</i>	address to store some bits of quotient at

Returns

remainder of floating-point division.

Exceptions

<i>FE_INVALID</i>	if <i>x</i> is infinite or <i>y</i> is 0 or if <i>x</i> or <i>y</i> is signaling NaN
-------------------	--

5.1.2.80 rint()

```
half half_float::rint (
    half arg) [inline]
```

Nearest integer using half's internal rounding mode. **See also:** Documentation for `std::rint`.

Parameters

<i>arg</i>	half expression to round
------------	--------------------------

Returns

nearest integer using default rounding mode

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_INEXACT</i>	if value had to be rounded

5.1.2.81 round()

```
half half_float::round (
    half arg) [inline]
```

Nearest integer. **See also:** Documentation for `std::round`.

Parameters

<i>arg</i>	half to round
------------	---------------

Returns

nearest integer, rounded away from zero in half-way cases

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_INEXACT</i>	if value had to be rounded

5.1.2.82 rsqrt()

```
half half_float::rsqrt (
    half arg) [inline]
```

Inverse square root. This function is exact to rounding for all rounding modes and thus generally more accurate than directly computing $1 / \text{sqrt}(arg)$ in half-precision, in addition to also being faster.

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

reciprocal of square root of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN and negative arguments
<i>FE_INEXACT</i>	according to rounding

5.1.2.83 scalbln()

```
half half_float::scalbln (
    half arg,
    long exp) [inline]
```

Multiply by power of two. This function is exact to rounding for all rounding modes.

See also: Documentation for [std::scalbln](#).

Parameters

<i>arg</i>	number to modify
<i>exp</i>	power of two to multiply with

Returns

arg multiplied by 2 raised to *exp*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.84 scalbn()

```
half half_float::scalbn (
    half arg,
    int exp) [inline]
```

Multiply by power of two. This function is exact to rounding for all rounding modes.

See also: Documentation for [std::scalbn](#).

Parameters

<i>arg</i>	number to modify
<i>exp</i>	power of two to multiply with

Returns

arg multiplied by 2 raised to *exp*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.85 signbit()

```
HALF_CONSTEXPR bool half_float::signbit (
    half arg) [inline]
```

Check sign. **See also:** Documentation for [std::signbit](#).

Parameters

<i>arg</i>	number to check
------------	-----------------

Return values

<i>true</i>	for negative number
<i>false</i>	for positive number

5.1.2.86 sin()

```
half half_float::sin (
    half arg) [inline]
```

Sine function. This function is exact to rounding for all rounding modes.

See also: Documentation for [std::sin](#).

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

sine value of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN or infinity
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.87 sincos()

```
void half_float::sincos (
    half arg,
    half * sin,
    half * cos) [inline]
```

Compute sine and cosine simultaneously. This returns the same results as [sin\(\)](#) and [cos\(\)](#) but is faster than calling each function individually.

This function is exact to rounding for all rounding modes.

Parameters

<i>arg</i>	function argument
<i>sin</i>	variable to take sine of <i>arg</i>
<i>cos</i>	variable to take cosine of <i>arg</i>

Exceptions

<i>FE_INVALID</i>	for signaling NaN or infinity
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.88 sinh()

```
half half_float::sinh (
    half arg) [inline]
```

Hyperbolic sine. This function is exact to rounding for all rounding modes.

See also: Documentation for [std::sinh](#).

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

hyperbolic sine value of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.89 sqrt()

```
half half_float::sqrt (
    half arg) [inline]
```

Square root. This function is exact to rounding for all rounding modes.

See also: Documentation for `std::sqrt`.

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

square root of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN and negative arguments
<i>FE_INEXACT</i>	according to rounding

5.1.2.90 tan()

```
half half_float::tan (
    half arg) [inline]
```

Tangent function. This function is exact to rounding for all rounding modes.

See also: Documentation for `std::tan`.

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

tangent value of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN or infinity
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.91 tanh()

```
half half_float::tanh (
    half arg) [inline]
```

Hyperbolic tangent. This function is exact to rounding for all rounding modes.

See also: Documentation for `std::tanh`.

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

hyperbolic tangent value of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.92 tgamma()

```
half half_float::tgamma (
    half arg) [inline]
```

Gamma function. This function may be 1 ULP off the correctly rounded exact result for any rounding mode in <0.25% of inputs.

See also: Documentation for `std::tgamma`.

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

gamma function value of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN, negative infinity or negative integer arguments
<i>FE_DIVBYZERO</i>	for 0
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

5.1.2.93 trunc()

```
half half_float::trunc (
    half arg) [inline]
```

Nearest integer not greater in magnitude than half value. **See also:** Documentation for `std::trunc`.

Parameters

<i>arg</i>	half to round
------------	---------------

Returns

nearest integer not greater in magnitude than *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_INEXACT</i>	if value had to be rounded

5.2 std Namespace Reference

Extensions to the C++ standard library.

Classes

- class [numeric_limits< half_float::half >](#)

5.2.1 Detailed Description

Extensions to the C++ standard library.

Chapter 6

Class Documentation

6.1 `half_float::detail::binary_t` Struct Reference

Tag type for binary construction.

```
#include <half.hpp>
```

6.1.1 Detailed Description

Tag type for binary construction.

The documentation for this struct was generated from the following file:

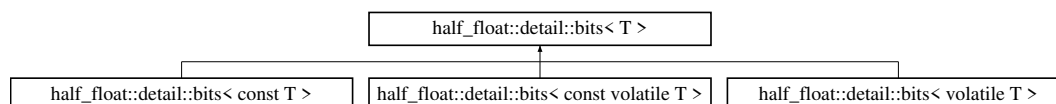
- [GL_Scene/half.hpp](#)

6.2 `half_float::detail::bits< T >` Struct Template Reference

Type traits for floating-point bits.

```
#include <half.hpp>
```

Inheritance diagram for `half_float::detail::bits< T >`:



Public Types

- typedef unsigned char **type**

6.2.1 Detailed Description

```
template<typename T>
struct half_float::detail::bits< T >
```

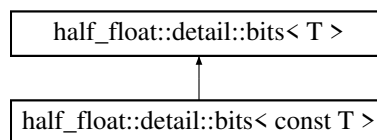
Type traits for floating-point bits.

The documentation for this struct was generated from the following file:

- [GL_Scene/half.hpp](#)

6.3 `half_float::detail::bits< const T >` Struct Template Reference

Inheritance diagram for `half_float::detail::bits< const T >`:



Public Types

- typedef unsigned char **type**

Public Types inherited from [half_float::detail::bits< T >](#)

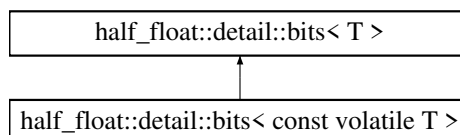
- typedef unsigned char **type**

The documentation for this struct was generated from the following file:

- [GL_Scene/half.hpp](#)

6.4 `half_float::detail::bits< const volatile T >` Struct Template Reference

Inheritance diagram for `half_float::detail::bits< const volatile T >`:



Public Types

- typedef unsigned char **type**

Public Types inherited from [half_float::detail::bits< T >](#)

- typedef unsigned char **type**

The documentation for this struct was generated from the following file:

- [GL_Scene/half.hpp](#)

6.5 half_float::detail::bits< double > Struct Reference

Unsigned integer of (at least) 64 bits width.

```
#include <half.hpp>
```

Public Types

- typedef unsigned long **type**
- typedef unsigned char **type**

6.5.1 Detailed Description

Unsigned integer of (at least) 64 bits width.

The documentation for this struct was generated from the following file:

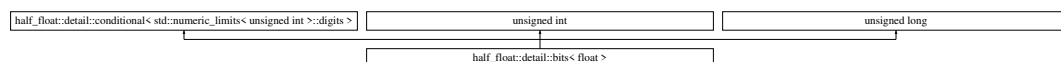
- [GL_Scene/half.hpp](#)

6.6 half_float::detail::bits< float > Struct Reference

Unsigned integer of (at least) 32 bits width.

```
#include <half.hpp>
```

Inheritance diagram for half_float::detail::bits< float >:



Public Types

- typedef unsigned char **type**

Public Types inherited from [half_float::detail::conditional< std::numeric_limits< unsigned int >::digits >](#)

- typedef T **type**

6.6.1 Detailed Description

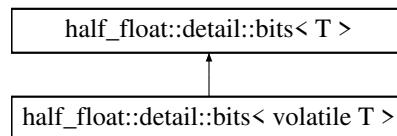
Unsigned integer of (at least) 32 bits width.

The documentation for this struct was generated from the following file:

- [GL_Scene/half.hpp](#)

6.7 `half_float::detail::bits< volatile T >` Struct Template Reference

Inheritance diagram for `half_float::detail::bits< volatile T >`:



Public Types

- typedef unsigned char **type**

Public Types inherited from [half_float::detail::bits< T >](#)

- typedef unsigned char **type**

The documentation for this struct was generated from the following file:

- [GL_Scene/half.hpp](#)

6.8 `half_float::detail::bool_type< bool >` Struct Template Reference

Helper for tag dispatching.

```
#include <half.hpp>
```

6.8.1 Detailed Description

```
template<bool>
struct half_float::detail::bool_type< bool >
```

Helper for tag dispatching.

The documentation for this struct was generated from the following file:

- [GL_Scene/half.hpp](#)

6.9 Camera Class Reference

Clase que gestiona la cámara en un entorno 3D, incluyendo el control de la posición, orientación y el movimiento de la cámara.

```
#include <Camera.hpp>
```

Public Member Functions

- [Camera](#) (glm::vec3 start_position, glm::vec3 up_direction, float start_yaw, float start_pitch)
Constructor de la cámara.
- glm::mat4 [get_view_matrix](#) () const
Obtiene la matriz de vista de la cámara.
- void [process_keyboard](#) (CameraMovement direction, float delta_time)
Procesa la entrada de teclado para mover la cámara.
- void [process_mouse_movement](#) (float x_offset, float y_offset, bool constraint_pitch=true)
Procesa el movimiento del ratón para rotar la cámara.

Public Attributes

- glm::vec3 [position](#)
Posición actual de la cámara.
- glm::vec3 [front](#)
Dirección hacia la cual está mirando la cámara.
- glm::vec3 [up](#)
Vectores de orientación de la cámara en el eje Y (arriba).
- glm::vec3 [right](#)
Vectores de la orientación de la cámara en el eje X (derecha).
- glm::vec3 [world_up](#)
Dirección "arriba" global.
- float [yaw](#)
Ángulo de orientación de la cámara alrededor del eje Y.
- float [pitch](#)
Ángulo de orientación de la cámara alrededor del eje X.
- float [movement_speed](#)
Velocidad de movimiento de la cámara.
- float [mouse_sensitivity](#)
Sensibilidad al movimiento del ratón.
- float [zoom](#)
Nivel de zoom de la cámara.

6.9.1 Detailed Description

Clase que gestiona la cámara en un entorno 3D, incluyendo el control de la posición, orientación y el movimiento de la cámara.

La clase [Camera](#) permite controlar la vista desde una cámara en 3D, proporcionando funcionalidades para mover la cámara en el espacio (adelante, atrás, izquierda, derecha, etc.), así como ajustar su orientación y zoom. Es comúnmente utilizada en aplicaciones gráficas en 3D, como videojuegos o simulaciones, donde se necesita un control interactivo sobre la vista de la escena.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 Camera()

```
Camera::Camera (
    glm::vec3 start_position,
    glm::vec3 up_direction,
    float start_yaw,
    float start_pitch)
```

Constructor de la cámara.

Inicializa una nueva cámara con la posición, dirección "arriba", yaw y pitch especificados.

Parameters

<i>start_position</i>	La posición inicial de la cámara en el espacio 3D.
<i>up_direction</i>	La dirección "arriba" de la cámara.
<i>start_yaw</i>	El ángulo de yaw inicial de la cámara.
<i>start_pitch</i>	El ángulo de pitch inicial de la cámara.

6.9.3 Member Function Documentation

6.9.3.1 get_view_matrix()

```
glm::mat4 Camera::get_view_matrix () const
```

Obtiene la matriz de vista de la cámara.

La matriz de vista se usa para transformar las coordenadas de la escena en relación con la posición y orientación de la cámara.

Returns

Una matriz 4x4 que representa la vista de la cámara.

6.9.3.2 process_keyboard()

```
void Camera::process_keyboard (
    CameraMovement direction,
    float delta_time)
```

Procesa la entrada de teclado para mover la cámara.

Cambia la posición de la cámara según la dirección especificada y el delta_time dado. El delta_time es usado para ajustar el movimiento en función del tiempo transcurrido.

Parameters

<i>direction</i>	La dirección en la que se desea mover la cámara (adelante, atrás, izquierda, derecha, etc.).
<i>delta_time</i>	El tiempo transcurrido desde el último fotograma, usado para controlar la velocidad.

6.9.3.3 process_mouse_movement()

```
void Camera::process_mouse_movement (
    float x_offset,
    float y_offset,
    bool constraint_pitch = true)
```

Procesa el movimiento del ratón para rotar la cámara.

Ajusta la orientación de la cámara en función de los movimientos del ratón. La sensibilidad de estos movimientos es controlada por el valor de `mouse_sensitivity`.

Parameters

<code>x_offset</code>	El cambio en la posición X del ratón.
<code>y_offset</code>	El cambio en la posición Y del ratón.
<code>constraint_pitch</code>	Si se debe restringir el ángulo de pitch para evitar una rotación excesiva.

6.9.4 Member Data Documentation

6.9.4.1 front

```
glm::vec3 Camera::front
```

Dirección hacia la cual está mirando la cámara.

Define la dirección en la que la cámara está mirando. Esto se utiliza para calcular la matriz de vista de la cámara.

6.9.4.2 mouse_sensitivity

```
float Camera::mouse_sensitivity
```

Sensibilidad al movimiento del ratón.

Controla cuánto se ajustan los ángulos de yaw y pitch cuando se mueve el ratón.

6.9.4.3 movement_speed

```
float Camera::movement_speed
```

Velocidad de movimiento de la cámara.

Define la rapidez con la que la cámara se mueve en función del `delta_time`.

6.9.4.4 pitch

```
float Camera::pitch
```

Ángulo de orientación de la cámara alrededor del eje X.

El ángulo de inclinación (pitch) controla la rotación de la cámara alrededor del eje horizontal.

6.9.4.5 position

```
glm::vec3 Camera::position
```

Posición actual de la cámara.

Esta es la posición de la cámara en el espacio 3D.

6.9.4.6 right

```
glm::vec3 Camera::right
```

Vectores de la orientación de la cámara en el eje X (derecha).

Define la dirección "derecha" de la cámara. Este vector es calculado en función del eje 'up' y 'front'.

6.9.4.7 up

```
glm::vec3 Camera::up
```

Vectores de orientación de la cámara en el eje Y (arriba).

Define la dirección del "arriba" de la cámara, utilizado para la orientación de la vista.

6.9.4.8 world_up

```
glm::vec3 Camera::world_up
```

Dirección "arriba" global.

Este es el vector global de "arriba" que se utiliza para la rotación de la cámara para mantener la orientación correcta de la cámara.

6.9.4.9 yaw

```
float Camera::yaw
```

Ángulo de orientación de la cámara alrededor del eje Y.

El ángulo de giro (yaw) se utiliza para girar la cámara alrededor del eje vertical.

6.9.4.10 zoom

```
float Camera::zoom
```

Nivel de zoom de la cámara.

Representa el zoom de la cámara, determinando el campo de visión (FOV).

The documentation for this class was generated from the following files:

- GL_Scene/Camera.hpp
- GL_Scene/Camera.cpp

6.10 half_float::detail::conditional< bool, T, typename > Struct Template Reference

Conditional type.

```
#include <half.hpp>
```

Public Types

- typedef T **type**

6.10.1 Detailed Description

```
template<bool, typename T, typename>  
struct half_float::detail::conditional< bool, T, typename >
```

Conditional type.

The documentation for this struct was generated from the following file:

- GL_Scene/[half.hpp](#)

6.11 half_float::detail::conditional< false, T, F > Struct Template Reference

Public Types

- typedef F **type**
- typedef T **type**

The documentation for this struct was generated from the following file:

- GL_Scene/[half.hpp](#)

6.12 Cube Class Reference

Clase que representa un cubo, heredando de la clase [Mesh](#).

```
#include <Cube.hpp>
```

6.12.1 Detailed Description

Clase que representa un cubo, heredando de la clase [Mesh](#).

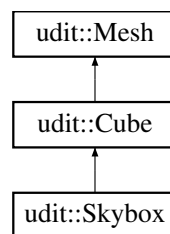
La clase [Cube](#) crea y gestiona un cubo 3D. Ofrece constructores para crear un cubo con un tamaño específico y con la opción de invertir las normales. Hereda de la clase [Mesh](#) y aprovecha sus funcionalidades para el procesamiento y renderizado del cubo en un entorno OpenGL.

The documentation for this class was generated from the following file:

- GL_Scene/Cube.hpp

6.13 udit::Cube Class Reference

Inheritance diagram for udit::Cube:



Public Member Functions

- [Cube](#) ()
Constructor por defecto.
- [Cube](#) (bool inverted)
Constructor con opción de invertir las normales.
- [Cube](#) (float size)
Constructor con tamaño especificado.
- [Cube](#) (float size, bool inverted)
Constructor con tamaño y opción de invertir las normales.

Public Member Functions inherited from [udit::Mesh](#)

- [Mesh](#) ()
Constructor por defecto.
- [Mesh](#) (std::string &path)
Constructor que carga una malla desde un archivo.
- virtual [~Mesh](#) ()
Destructor de la clase.
- virtual void [translate](#) (glm::vec3 translation)
Realiza una traslación de la malla.
- virtual void [rotate](#) (glm::vec3 rotation, float angle)
Rota la malla.
- virtual void [scale](#) (glm::vec3 scale)

- Escala la malla.*
- virtual void [update](#) ()
- Actualiza la malla.*
- virtual void [render](#) (glm::mat4 view_matrix)
- Renderiza la malla.*
- virtual void [resize](#) (glm::mat4 projection_matrix)
- Ajusta la matriz de proyección.*
- virtual void [set_shader](#) (std::shared_ptr< [udit::Shader](#) > shader)
- Asocia un shader a la malla.*
- GLuint [get_shader_program_id](#) () const
- Obtiene el ID del programa del shader asociado.*
- std::vector< GLint > [get_shader_matrix_ids](#) ()
- Obtiene los IDs de las matrices del shader asociadas a la malla.*
- glm::mat4 [get_model_view_matrix](#) () const
- Obtiene la matriz de transformación del modelo.*
- void [set_model_view_matrix](#) (glm::mat4 matrix)
- Establece la matriz de transformación del modelo.*
- void [set_mesh_type](#) (MeshType type)
- Establece el tipo de malla.*

Additional Inherited Members

Static Public Member Functions inherited from [udit::Mesh](#)

- static std::shared_ptr< [Mesh](#) > [make_mesh](#) (MeshType type, const std::string &path="")
- Crea una malla de un tipo específico.*

Protected Member Functions inherited from [udit::Mesh](#)

- void [create_mesh](#) (std::string mesh_name="")
- Crea los VBOs y el VAO necesarios para la malla.*

Protected Attributes inherited from [udit::Mesh](#)

- std::vector< glm::vec3 > **coordinates**
- Vectores que almacenan las coordenadas de los vértices, colores, normales, índices y coordenadas de textura.*
- std::vector< glm::vec3 > **colors**
- std::vector< glm::vec3 > **normals**
- std::vector< GLuint > **indices**
- std::vector< glm::vec2 > **texture_uvs**
- GLsizei **number_of_vertices**
- Número total de vértices de la malla.*

6.13.1 Constructor & Destructor Documentation

6.13.1.1 Cube() [1/4]

`Cube::Cube ()`

Constructor por defecto.

Este constructor crea un cubo con un tamaño predeterminado y sin invertir las normales.

6.13.1.2 Cube() [2/4]

```
Cube::Cube (
    bool inverted)
```

Constructor con opción de invertir las normales.

Este constructor crea un cubo con un tamaño predeterminado. La opción de invertir las normales puede ser útil para efectos especiales como la renderización por dentro del cubo.

Parameters

<i>inverted</i>	Si es <code>true</code> , las normales del cubo se invierten.
-----------------	---

6.13.1.3 Cube() [3/4]

```
Cube::Cube (
    float size)
```

Constructor con tamaño especificado.

Este constructor crea un cubo con un tamaño determinado y sin invertir las normales.

Parameters

<i>size</i>	El tamaño de los lados del cubo.
-------------	----------------------------------

6.13.1.4 Cube() [4/4]

```
Cube::Cube (
    float size,
    bool inverted)
```

Constructor con tamaño y opción de invertir las normales.

Este constructor permite crear un cubo de cualquier tamaño, con la opción de invertir las normales. La inversión de las normales puede ser útil para representar el cubo desde dentro.

Parameters

<i>size</i>	El tamaño de los lados del cubo.
<i>inverted</i>	Si es <code>true</code> , las normales del cubo se invierten.

The documentation for this class was generated from the following files:

- GL_Scene/Cube.hpp
- GL_Scene/Cube.cpp

6.14 EventHandler Class Reference

Clase que maneja los eventos de entrada (teclado, ratón) en la escena.

```
#include <EventHandler.hpp>
```

Public Member Functions

- [EventHandler](#) ([Camera](#) &camera)
Constructor que inicializa el [EventHandler](#) con una referencia a la cámara.
- void [handle_events](#) (bool &running, float delta_time)
Procesa los eventos de entrada y actualiza el estado de la cámara.

6.14.1 Detailed Description

Clase que maneja los eventos de entrada (teclado, ratón) en la escena.

La clase [EventHandler](#) es responsable de gestionar los eventos de entrada provenientes de dispositivos como el teclado y el ratón. Se encarga de procesar dichos eventos y actualiza la cámara en consecuencia, permitiendo la navegación a través de la escena 3D.

6.14.2 Constructor & Destructor Documentation

6.14.2.1 EventHandler()

```
EventHandler::EventHandler (
    Camera & camera) [inline]
```

Constructor que inicializa el [EventHandler](#) con una referencia a la cámara.

Este constructor inicializa el manejador de eventos con la cámara a la que se le enviarán las actualizaciones. También establece valores predeterminados para el seguimiento del ratón.

Parameters

<i>camera</i>	La cámara que se actualizará en respuesta a los eventos.
---------------	--

6.14.3 Member Function Documentation

6.14.3.1 handle_events()

```
void EventHandler::handle_events (
    bool & running,
    float delta_time)
```

Procesa los eventos de entrada y actualiza el estado de la cámara.

Esta función maneja los eventos generados por el sistema (teclado, ratón) y, dependiendo del tipo de evento, realiza las actualizaciones necesarias en la cámara, como moverla o rotarla. Esta función debe ser llamada en cada ciclo del bucle de renderizado.

Parameters

<i>running</i>	Un parámetro que indica si el bucle de la aplicación sigue en ejecución. Si se establece a <code>false</code> , el bucle terminará.
<i>delta_time</i>	El tiempo transcurrido entre el fotograma actual y el anterior. Se utiliza para asegurar un movimiento suave de la cámara.

The documentation for this class was generated from the following files:

- GL_Scene/EventHandler.hpp
- GL_Scene/EventHandler.cpp

6.15 half_float::detail::f31 Struct Reference

Class for 1.31 unsigned floating-point computation.

```
#include <half.hpp>
```

Public Member Functions

- HALF_CONSTEXPR [f31](#) ([uint32](#) mant, int e)
- [f31](#) (unsigned int [abs](#))

Public Attributes

- [uint32](#) **m**
mantissa as 1.31.
- int **exp**
exponent.

Friends

- [f31 operator+](#) ([f31](#) a, [f31](#) b)
- [f31 operator-](#) ([f31](#) a, [f31](#) b)
- [f31 operator*](#) ([f31](#) a, [f31](#) b)
- [f31 operator/](#) ([f31](#) a, [f31](#) b)

6.15.1 Detailed Description

Class for 1.31 unsigned floating-point computation.

6.15.2 Constructor & Destructor Documentation

6.15.2.1 f31() [1/2]

```
HALF_CONSTEXPR half_float::detail::f31::f31 (
    uint32 mant,
    int e) [inline]
```

Constructor.

Parameters

<i>mant</i>	mantissa as 1.31
<i>e</i>	exponent

6.15.2.2 f31() [2/2]

```
half_float::detail::f31::f31 (  
    unsigned int abs) [inline]
```

Constructor.

Parameters

<i>abs</i>	unsigned half-precision value
------------	-------------------------------

6.15.3 Friends And Related Symbol Documentation

6.15.3.1 operator*

```
f31 operator* (  
    f31 a,  
    f31 b) [friend]
```

Multiplication operator.

Parameters

<i>a</i>	first operand
<i>b</i>	second operand

Returns

$a * b$

6.15.3.2 operator+

```
f31 operator+ (  
    f31 a,  
    f31 b) [friend]
```

Addition operator.

Parameters

<i>a</i>	first operand
<i>b</i>	second operand

Returns

$a + b$

6.15.3.3 operator-

```
f31 operator- (
    f31 a,
    f31 b) [friend]
```

Subtraction operator.

Parameters

<i>a</i>	first operand
<i>b</i>	second operand

Returns

$a - b$

6.15.3.4 operator/

```
f31 operator/ (
    f31 a,
    f31 b) [friend]
```

Division operator.

Parameters

<i>a</i>	first operand
<i>b</i>	second operand

Returns

a / b

The documentation for this struct was generated from the following file:

- [GL_Scene/half.hpp](#)

6.16 half_float::half Class Reference

```
#include <half.hpp>
```

Public Member Functions

Construction and assignment

- HALF_CONSTEXPR [half](#) () HALF_NOEXCEPT
- [half](#) (float rhs)
- [operator float](#) () const
- [half](#) & [operator=](#) (float rhs)

Arithmetic updates

- [half](#) & [operator+=](#) ([half](#) rhs)
- [half](#) & [operator-=](#) ([half](#) rhs)
- [half](#) & [operator*=](#) ([half](#) rhs)
- [half](#) & [operator/=](#) ([half](#) rhs)
- [half](#) & [operator+=](#) (float rhs)
- [half](#) & [operator-=](#) (float rhs)
- [half](#) & [operator*=](#) (float rhs)
- [half](#) & [operator/=](#) (float rhs)

Increment and decrement

- [half](#) & [operator++](#) ()
- [half](#) & [operator--](#) ()
- [half](#) [operator++](#) (int)
- [half](#) [operator--](#) (int)

Friends

- template<typename, typename, std::float_round_style>
struct **detail::half_caster**
- class **std::numeric_limits< half >**
- HALF_CONSTEXPR_NOERR bool [operator==](#) ([half](#) x, [half](#) y)
- HALF_CONSTEXPR_NOERR bool [operator!=](#) ([half](#) x, [half](#) y)
- HALF_CONSTEXPR_NOERR bool [operator<](#) ([half](#) x, [half](#) y)
- HALF_CONSTEXPR_NOERR bool [operator>](#) ([half](#) x, [half](#) y)
- HALF_CONSTEXPR_NOERR bool [operator<=](#) ([half](#) x, [half](#) y)
- HALF_CONSTEXPR_NOERR bool [operator>=](#) ([half](#) x, [half](#) y)
- HALF_CONSTEXPR [half](#) [operator-](#) ([half](#) arg)
- [half](#) [operator+](#) ([half](#) x, [half](#) y)
- [half](#) [operator-](#) ([half](#) x, [half](#) y)
- [half](#) [operator*](#) ([half](#) x, [half](#) y)
- [half](#) [operator/](#) ([half](#) x, [half](#) y)
- template<typename charT, typename traits>
std::basic_ostream< charT, traits > & [operator<<](#) (std::basic_ostream< charT, traits > &out, [half](#) arg)
- template<typename charT, typename traits>
std::basic_istream< charT, traits > & [operator>>](#) (std::basic_istream< charT, traits > &in, [half](#) &arg)
- HALF_CONSTEXPR [half](#) [fabs](#) ([half](#) arg)
- [half](#) [fmod](#) ([half](#) x, [half](#) y)
- [half](#) [remainder](#) ([half](#) x, [half](#) y)
- [half](#) [remquo](#) ([half](#) x, [half](#) y, int *quo)
- [half](#) [fma](#) ([half](#) x, [half](#) y, [half](#) z)
- HALF_CONSTEXPR_NOERR [half](#) [fmax](#) ([half](#) x, [half](#) y)
- HALF_CONSTEXPR_NOERR [half](#) [fmin](#) ([half](#) x, [half](#) y)
- [half](#) [fdim](#) ([half](#) x, [half](#) y)
- [half](#) [nanh](#) (const char *arg)

- [half exp](#) ([half](#) arg)
- [half exp2](#) ([half](#) arg)
- [half expm1](#) ([half](#) arg)
- [half log](#) ([half](#) arg)
- [half log10](#) ([half](#) arg)
- [half log2](#) ([half](#) arg)
- [half log1p](#) ([half](#) arg)
- [half sqrt](#) ([half](#) arg)
- [half rsqrt](#) ([half](#) arg)
- [half cbrt](#) ([half](#) arg)
- [half hypot](#) ([half](#) x, [half](#) y)
- [half hypot](#) ([half](#) x, [half](#) y, [half](#) z)
- [half pow](#) ([half](#) x, [half](#) y)
- void [sincos](#) ([half](#) arg, [half](#) *sin, [half](#) *cos)
- [half sin](#) ([half](#) arg)
- [half cos](#) ([half](#) arg)
- [half tan](#) ([half](#) arg)
- [half asin](#) ([half](#) arg)
- [half acos](#) ([half](#) arg)
- [half atan](#) ([half](#) arg)
- [half atan2](#) ([half](#) y, [half](#) x)
- [half sinh](#) ([half](#) arg)
- [half cosh](#) ([half](#) arg)
- [half tanh](#) ([half](#) arg)
- [half asinh](#) ([half](#) arg)
- [half acosh](#) ([half](#) arg)
- [half atanh](#) ([half](#) arg)
- [half erf](#) ([half](#) arg)
- [half erfc](#) ([half](#) arg)
- [half lgamma](#) ([half](#) arg)
- [half tgamma](#) ([half](#) arg)
- [half ceil](#) ([half](#) arg)
- [half floor](#) ([half](#) arg)
- [half trunc](#) ([half](#) arg)
- [half round](#) ([half](#) arg)
- long [lround](#) ([half](#) arg)
- [half rint](#) ([half](#) arg)
- long [lrint](#) ([half](#) arg)
- [half nearbyint](#) ([half](#) arg)
- [half frexp](#) ([half](#) arg, int *exp)
- [half scalbln](#) ([half](#) arg, long exp)
- [half modf](#) ([half](#) arg, [half](#) *iptr)
- int [ilogb](#) ([half](#) arg)
- [half logb](#) ([half](#) arg)
- [half nextafter](#) ([half](#) from, [half](#) to)
- [half nexttoward](#) ([half](#) from, long double to)
- HALF_CONSTEXPR [half copysign](#) ([half](#) x, [half](#) y)
- HALF_CONSTEXPR int [fpclassify](#) ([half](#) arg)
- HALF_CONSTEXPR bool [isfinite](#) ([half](#) arg)
- HALF_CONSTEXPR bool [isinf](#) ([half](#) arg)
- HALF_CONSTEXPR bool [isnan](#) ([half](#) arg)
- HALF_CONSTEXPR bool [isnormal](#) ([half](#) arg)
- HALF_CONSTEXPR bool [signbit](#) ([half](#) arg)
- HALF_CONSTEXPR bool [isgreater](#) ([half](#) x, [half](#) y)
- HALF_CONSTEXPR bool [isgreaterequal](#) ([half](#) x, [half](#) y)
- HALF_CONSTEXPR bool [isless](#) ([half](#) x, [half](#) y)
- HALF_CONSTEXPR bool [islessequal](#) ([half](#) x, [half](#) y)
- HALF_CONSTEXPR bool [islessgreater](#) ([half](#) x, [half](#) y)

6.16.1 Detailed Description

Half-precision floating-point type. This class implements an IEEE-conformant half-precision floating-point type with the usual arithmetic operators and conversions. It is implicitly convertible to single-precision floating-point, which makes arithmetic expressions and functions with mixed-type operands to be of the most precise operand type.

According to the C++98/03 definition, the half type is not a POD type. But according to C++11's less strict and extended definitions it is both a standard layout type and a trivially copyable type (even if not a POD type), which means it can be standard-conformantly copied using raw binary copies. But in this context some more words about the actual size of the type. Although the half is representing an IEEE 16-bit type, it does not necessarily have to be of exactly 16-bits size. But on any reasonable implementation the actual binary representation of this type will most probably not involve any additional "magic" or padding beyond the simple binary representation of the underlying 16-bit IEEE number, even if not strictly guaranteed by the standard. But even then it only has an actual size of 16 bits if your C++ implementation supports an unsigned integer type of exactly 16 bits width. But this should be the case on nearly any reasonable platform.

So if your C++ implementation is not totally exotic or imposes special alignment requirements, it is a reasonable assumption that the data of a half is just comprised of the 2 bytes of the underlying IEEE representation.

6.16.2 Constructor & Destructor Documentation

6.16.2.1 half() [1/2]

```
HALF_CONSTEXPR half_float::half::half () [inline]
```

Default constructor. This initializes the half to 0. Although this does not match the builtin types' default-initialization semantics and may be less efficient than no initialization, it is needed to provide proper value-initialization semantics.

6.16.2.2 half() [2/2]

```
half_float::half::half (
    float rhs) [inline], [explicit]
```

Conversion constructor.

Parameters

<i>rhs</i>	float to convert
------------	------------------

Exceptions

<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding
--	-----------------------

6.16.3 Member Function Documentation

6.16.3.1 operator float()

```
half_float::half::operator float () const [inline]
```

Conversion to single-precision.

Returns

single precision value representing expression value

6.16.3.2 operator*=() [1/2]

```
half & half_float::half::operator*= (
    float rhs) [inline]
```

Arithmetic assignment.

Parameters

<i>rhs</i>	single-precision value to multiply with
------------	---

Returns

reference to this half

Exceptions

FE_{\leftrightarrow} _...	according to operator=()
--------------------------------	--

6.16.3.3 operator*=() [2/2]

```
half & half_float::half::operator*= (
    half rhs) [inline]
```

Arithmetic assignment.

Template Parameters

<i>T</i>	type of concrete half expression
----------	----------------------------------

Parameters

<i>rhs</i>	half expression to multiply with
------------	----------------------------------

Returns

reference to this half

Exceptions

FE_{\leftrightarrow} _...	according to operator*(half,half)
--------------------------------	---

6.16.3.4 operator++() [1/2]

```
half & half_float::half::operator++ () [inline]
```

Prefix increment.

Returns

incremented half value

Exceptions

<i>FE</i> ↔	according to operator+(half,half)
...	

6.16.3.5 operator++() [2/2]

```
half half_float::half::operator++ (
    int ) [inline]
```

Postfix increment.

Returns

non-incremented half value

Exceptions

<i>FE</i> ↔	according to operator+(half,half)
...	

6.16.3.6 operator+=() [1/2]

```
half & half_float::half::operator+= (
    float rhs) [inline]
```

Arithmetic assignment.

Parameters

<i>rhs</i>	single-precision value to add
------------	-------------------------------

Returns

reference to this half

Exceptions

<i>FE</i> ↔	according to operator=()
...	

6.16.3.7 operator+=() [2/2]

```
half & half_float::half::operator+= (
    half rhs) [inline]
```

Arithmetic assignment.

Template Parameters

<i>T</i>	type of concrete half expression
----------	----------------------------------

Parameters

<i>rhs</i>	half expression to add
------------	------------------------

Returns

reference to this half

Exceptions

<i>FE</i> ↔ _...	according to operator+(half,half)
---------------------	---

6.16.3.8 operator--() [1/2]

```
half & half_float::half::operator-- () [inline]
```

Prefix decrement.

Returns

decremented half value

Exceptions

<i>FE</i> ↔ _...	according to operator-(half,half)
---------------------	---

6.16.3.9 operator--() [2/2]

```
half half_float::half::operator-- (
    int ) [inline]
```

Postfix decrement.

Returns

non-decremented half value

Exceptions

<i>FE</i> ↔ _...	according to operator-(half,half)
---------------------	---

6.16.3.10 operator-=() [1/2]

```
half & half_float::half::operator-= (
    float rhs) [inline]
```

Arithmetic assignment.

Parameters

<i>rhs</i>	single-precision value to subtract
------------	------------------------------------

Returns

reference to this half

Exceptions

<i>FE</i> ↔ _...	according to operator=()
---------------------	--

6.16.3.11 operator-=() [2/2]

```
half & half_float::half::operator-= (
    half rhs) [inline]
```

Arithmetic assignment.

Template Parameters

<i>T</i>	type of concrete half expression
----------	----------------------------------

Parameters

<i>rhs</i>	half expression to subtract
------------	-----------------------------

Returns

reference to this half

Exceptions

<i>FE</i> ↔ _...	according to operator-(half,half)
---------------------	---

6.16.3.12 operator/=() [1/2]

```
half & half_float::half::operator/= (
    float rhs) [inline]
```

Arithmetic assignment.

Parameters

<i>rhs</i>	single-precision value to divide by
------------	-------------------------------------

Returns

reference to this half

Exceptions

<i>FE</i> ↔	according to operator=()
—...	

6.16.3.13 operator/=() [2/2]

```
half & half_float::half::operator/= (
    half rhs) [inline]
```

Arithmetic assignment.

Template Parameters

<i>T</i>	type of concrete half expression
----------	----------------------------------

Parameters

<i>rhs</i>	half expression to divide by
------------	------------------------------

Returns

reference to this half

Exceptions

<i>FE</i> ↔	according to operator/(half,half)
—...	

6.16.3.14 operator=()

```
half & half_float::half::operator= (
    float rhs) [inline]
```

Assignment operator.

Parameters

<i>rhs</i>	single-precision value to copy from
------------	-------------------------------------

Returns

reference to this half

Exceptions

<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding
--	-----------------------

6.16.4 Friends And Related Symbol Documentation

6.16.4.1 acos

```
half acos (
    half arg) [friend]
```

Arc cosine function. This function is exact to rounding for all rounding modes.

See also: Documentation for `std::acos`.

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

arc cosine value of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN or if $\text{abs}(arg) > 1$
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

6.16.4.2 acosh

```
half acosh (
    half arg) [friend]
```

Hyperbolic area cosine. This function is exact to rounding for all rounding modes.

See also: Documentation for `std::acosh`.

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

area cosine value of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN or arguments < 1
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

6.16.4.3 asin

```
half asin (
    half arg) [friend]
```

Arc sine. This function is exact to rounding for all rounding modes.

See also: Documentation for `std::asin`.

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

arc sine value of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN or if $\text{abs}(arg) > 1$
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

6.16.4.4 asinh

```
half asinh (
    half arg) [friend]
```

Hyperbolic area sine. This function is exact to rounding for all rounding modes.

See also: Documentation for `std::asinh`.

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

area sine value of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

6.16.4.5 atan

```
half atan (
    half arg) [friend]
```

Arc tangent function. This function is exact to rounding for all rounding modes.

See also: Documentation for `std::atan`.

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

arc tangent value of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

6.16.4.6 atan2

```
half atan2 (
    half y,
    half x) [friend]
```

Arc tangent function. This function may be 1 ULP off the correctly rounded exact result in $\sim 0.005\%$ of inputs for `std::round_to_nearest`, in $\sim 0.1\%$ of inputs for `std::round_toward_zero` and in $\sim 0.02\%$ of inputs for any other rounding mode.

See also: Documentation for `std::atan2`.

Parameters

<i>y</i>	numerator
<i>x</i>	denominator

Returns

arc tangent value

Exceptions

<i>FE_INVALID</i>	if <i>x</i> or <i>y</i> is signaling NaN
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

6.16.4.7 atanh

```
half atanh (
    half arg) [friend]
```

Hyperbolic area tangent. This function is exact to rounding for all rounding modes.

See also: Documentation for `std::atanh`.

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

area tangent value of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN or if $\text{abs}(arg) > 1$
<i>FE_DIVBYZERO</i>	for +/-1
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

6.16.4.8 cbrt

```
half cbrt (
    half arg) [friend]
```

Cubic root. This function is exact to rounding for all rounding modes.

See also: Documentation for `std::cbrt`.

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

cubic root of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_INEXACT</i>	according to rounding

6.16.4.9 ceil

```
half ceil (
    half arg) [friend]
```

Nearest integer not less than half value. **See also:** Documentation for `std::ceil`.

Parameters

<i>arg</i>	half to round
------------	---------------

Returns

nearest integer not less than *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_INEXACT</i>	if value had to be rounded

6.16.4.10 copysign

```
HALF_CONSTEXPR half copysign (
    half x,
    half y) [friend]
```

Take sign. **See also:** Documentation for [std::copysign](#).

Parameters

<i>x</i>	value to change sign for
<i>y</i>	value to take sign from

Returns

value equal to *x* in magnitude and to *y* in sign

6.16.4.11 cos

```
half cos (
    half arg) [friend]
```

Cosine function. This function is exact to rounding for all rounding modes.

See also: Documentation for [std::cos](#).

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

cosine value of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN or infinity
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

6.16.4.12 cosh

```
half cosh (
    half arg) [friend]
```

Hyperbolic cosine. This function is exact to rounding for all rounding modes.

See also: Documentation for [std::cosh](#).

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

hyperbolic cosine value of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

6.16.4.13 erf

```
half erf (
    half arg) [friend]
```

Error function. This function may be 1 ULP off the correctly rounded exact result for any rounding mode in <0.5% of inputs.

See also: Documentation for [std::erf](#).

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

error function value of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

6.16.4.14 erfc

```
half erfc (
    half arg) [friend]
```

Complementary error function. This function may be 1 ULP off the correctly rounded exact result for any rounding mode in <0.5% of inputs.

See also: Documentation for [std::erfc](#).

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

1 minus error function value of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

6.16.4.15 exp

```
half exp (
    half arg) [friend]
```

Exponential function. This function is exact to rounding for all rounding modes.

See also: Documentation for [std::exp](#).

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

e raised to *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

6.16.4.16 exp2

```
half exp2 (
    half arg) [friend]
```

Binary exponential. This function is exact to rounding for all rounding modes.

See also: Documentation for [std::exp2](#).

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

2 raised to *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

6.16.4.17 expm1

```
half expm1 (
    half arg) [friend]
```

Exponential minus one. This function may be 1 ULP off the correctly rounded exact result in <0.05% of inputs for `std::round_to_nearest` and in <1% of inputs for any other rounding mode.

See also: Documentation for `std::expm1`.

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

e raised to *arg* and subtracted by 1

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

6.16.4.18 fabs

```
HALF_CONSTEXPR half fabs (
    half arg) [friend]
```

Absolute value. **See also:** Documentation for `std::fabs`.

Parameters

<i>arg</i>	operand
------------	---------

Returns

absolute value of *arg*

6.16.4.19 fdim

```
half fdim (
    half x,
    half y) [friend]
```

Positive difference. This function is exact to rounding for all rounding modes.

See also: Documentation for `std::fdim`.

Parameters

<i>x</i>	first operand
<i>y</i>	second operand

Returns

$x - y$ or 0 if difference negative

Exceptions

<i>FE_↔</i>	according to operator-(half,half)
...	

6.16.4.20 floor

```
half floor (
    half arg) [friend]
```

Nearest integer not greater than half value. **See also:** Documentation for [std::floor](#).

Parameters

<i>arg</i>	half to round
------------	---------------

Returns

nearest integer not greater than *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_INEXACT</i>	if value had to be rounded

6.16.4.21 fma

```
half fma (
    half x,
    half y,
    half z) [friend]
```

Fused multiply add. This function is exact to rounding for all rounding modes.

See also: Documentation for [std::fma](#).

Parameters

<i>x</i>	first operand
<i>y</i>	second operand
<i>z</i>	third operand

Returns

$(x * y) + z$ rounded as one operation.

Exceptions

<i>FE_INVALID</i>	according to operator*() and operator+() unless any argument is a quiet NaN and no argument is a signaling NaN
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding the final addition

6.16.4.22 fmax

```
HALF_CONSTEXPR_NOERR half fmax (
    half x,
    half y) [friend]
```

Maximum of half expressions. **See also:** Documentation for [std::fmax](#).

Parameters

<i>x</i>	first operand
<i>y</i>	second operand

Returns

maximum of operands, ignoring quiet NaNs

Exceptions

<i>FE_INVALID</i>	if <i>x</i> or <i>y</i> is signaling NaN
-------------------	--

6.16.4.23 fmin

```
HALF_CONSTEXPR_NOERR half fmin (
    half x,
    half y) [friend]
```

Minimum of half expressions. **See also:** Documentation for [std::fmin](#).

Parameters

<i>x</i>	first operand
<i>y</i>	second operand

Returns

minimum of operands, ignoring quiet NaNs

Exceptions

<i>FE_INVALID</i>	if <i>x</i> or <i>y</i> is signaling NaN
-------------------	--

6.16.4.24 fmod

```
half fmod (  
    half x,  
    half y) [friend]
```

Remainder of division. **See also:** Documentation for [std::fmod](#).

Parameters

<i>x</i>	first operand
<i>y</i>	second operand

Returns

remainder of floating-point division.

Exceptions

<i>FE_INVALID</i>	if <i>x</i> is infinite or <i>y</i> is 0 or if <i>x</i> or <i>y</i> is signaling NaN
-------------------	--

6.16.4.25 fpclassify

```
HALF_CONSTEXPR int fpclassify (  
    half arg) [friend]
```

Classify floating-point value. **See also:** Documentation for [std::fpclassify](#).

Parameters

<i>arg</i>	number to classify
------------	--------------------

Return values

<i>FP_ZERO</i>	for positive and negative zero
<i>FP_SUBNORMAL</i>	for subnormal numbers
<i>FP_INFINITY</i>	for positive and negative infinity
<i>FP_NAN</i>	for NaNs
<i>FP_NORMAL</i>	for all other (normal) values

6.16.4.26 frexp

```
half frexp (
    half arg,
    int * exp) [friend]
```

Decompress floating-point number. **See also:** Documentation for `std::frexp`.

Parameters

<i>arg</i>	number to decompress
<i>exp</i>	address to store exponent at

Returns

significant in range [0.5, 1)

Exceptions

<i>FE_INVALID</i>	for signaling NaN
-------------------	-------------------

6.16.4.27 hypot [1/2]

```
half hypot (
    half x,
    half y) [friend]
```

Hypotenuse function. This function is exact to rounding for all rounding modes.

See also: Documentation for `std::hypot`.

Parameters

<i>x</i>	first argument
<i>y</i>	second argument

Returns

square root of sum of squares without internal over- or underflows

Exceptions

<i>FE_INVALID</i>	if <i>x</i> or <i>y</i> is signaling NaN
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding of the final square root

6.16.4.28 hypot [2/2]

```
half hypot (
    half x,
    half y,
    half z) [friend]
```

Hypotenuse function. This function is exact to rounding for all rounding modes.

See also: Documentation for `std::hypot`.

Parameters

<i>x</i>	first argument
<i>y</i>	second argument
<i>z</i>	third argument

Returns

square root of sum of squares without internal over- or underflows

Exceptions

<i>FE_INVALID</i>	if <i>x</i> , <i>y</i> or <i>z</i> is signaling NaN
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding of the final square root

6.16.4.29 ilogb

```
int ilogb (
    half arg) [friend]
```

Extract exponent. **See also:** Documentation for `std::ilogb`.

Parameters

<i>arg</i>	number to query
------------	-----------------

Returns

floating-point exponent

Return values

<i>FP_ILOGB0</i>	for zero
<i>FP_ILOGBNAN</i>	for NaN
<i>INT_MAX</i>	for infinity

Exceptions

<i>FE_INVALID</i>	for 0 or infinite values
-------------------	--------------------------

6.16.4.30 isfinite

```
HALF_CONSTEXPR bool isfinite (
    half arg) [friend]
```

Check if finite number. **See also:** Documentation for [std::isfinite](#).

Parameters

<i>arg</i>	number to check
------------	-----------------

Return values

<i>true</i>	if neither infinity nor NaN
<i>false</i>	else

6.16.4.31 isgreater

```
HALF_CONSTEXPR bool isgreater (
    half x,
    half y) [friend]
```

Quiet comparison for greater than. **See also:** Documentation for [std::isgreater](#).

Parameters

<i>x</i>	first operand
<i>y</i>	second operand

Return values

<i>true</i>	if <i>x</i> greater than <i>y</i>
<i>false</i>	else

6.16.4.32 isgreaterequal

```
HALF_CONSTEXPR bool isgreaterequal (
    half x,
    half y) [friend]
```

Quiet comparison for greater equal. **See also:** Documentation for [std::isgreaterequal](#).

Parameters

<i>x</i>	first operand
<i>y</i>	second operand

Return values

<i>true</i>	if <i>x</i> greater equal <i>y</i>
<i>false</i>	else

6.16.4.33 isinf

```
HALF_CONSTEXPR bool isinf (  
    half arg) [friend]
```

Check for infinity. **See also:** Documentation for [std::isinf](#).

Parameters

<i>arg</i>	number to check
------------	-----------------

Return values

<i>true</i>	for positive or negative infinity
<i>false</i>	else

6.16.4.34 isless

```
HALF_CONSTEXPR bool isless (  
    half x,  
    half y) [friend]
```

Quiet comparison for less than. **See also:** Documentation for [std::isless](#).

Parameters

<i>x</i>	first operand
<i>y</i>	second operand

Return values

<i>true</i>	if <i>x</i> less than <i>y</i>
<i>false</i>	else

6.16.4.35 islessequal

```
HALF_CONSTEXPR bool islessequal (  
    half x,  
    half y) [friend]
```

Quiet comparison for less equal. **See also:** Documentation for [std::islessequal](#).

Parameters

<i>x</i>	first operand
<i>y</i>	second operand

Return values

<i>true</i>	if <i>x</i> less equal <i>y</i>
<i>false</i>	else

6.16.4.36 islessgreater

```
HALF_CONSTEXPR bool islessgreater (
    half x,
    half y) [friend]
```

Quiet comarison for less or greater. **See also:** Documentation for [std::islessgreater](#).

Parameters

<i>x</i>	first operand
<i>y</i>	second operand

Return values

<i>true</i>	if either less or greater
<i>false</i>	else

6.16.4.37 isnan

```
HALF_CONSTEXPR bool isnan (
    half arg) [friend]
```

Check for NaN. **See also:** Documentation for [std::isnan](#).

Parameters

<i>arg</i>	number to check
------------	-----------------

Return values

<i>true</i>	for NaNs
<i>false</i>	else

6.16.4.38 isnormal

```
HALF_CONSTEXPR bool isnormal (
    half arg) [friend]
```

Check if normal number. **See also:** Documentation for [std::isnormal](#).

Parameters

<i>arg</i>	number to check
------------	-----------------

Return values

<i>true</i>	if normal number
<i>false</i>	if either subnormal, zero, infinity or NaN

6.16.4.39 lgamma

```
half lgamma (
    half arg) [friend]
```

Natural logarithm of gamma function. This function may be 1 ULP off the correctly rounded exact result for any rounding mode in $\sim 0.025\%$ of inputs.

See also: Documentation for `std::lgamma`.

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

natural logarithm of gamma function for *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_DIVBYZERO</i>	for 0 or negative integer arguments
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

6.16.4.40 log

```
half log (
    half arg) [friend]
```

Natural logarithm. This function is exact to rounding for all rounding modes.

See also: Documentation for `std::log`.

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

logarithm of *arg* to base e

Exceptions

<i>FE_INVALID</i>	for signaling NaN or negative argument
<i>FE_DIVBYZERO</i>	for 0
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

6.16.4.41 log10

```
half log10 (
    half arg) [friend]
```

Common logarithm. This function is exact to rounding for all rounding modes.

See also: Documentation for `std::log10`.

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

logarithm of *arg* to base 10

Exceptions

<i>FE_INVALID</i>	for signaling NaN or negative argument
<i>FE_DIVBYZERO</i>	for 0
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

6.16.4.42 log1p

```
half log1p (
    half arg) [friend]
```

Natural logarithm plus one. This function may be 1 ULP off the correctly rounded exact result in <0.05% of inputs for `std::round_to_nearest` and in ~1% of inputs for any other rounding mode.

See also: Documentation for `std::log1p`.

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

logarithm of *arg* plus 1 to base e

Exceptions

<i>FE_INVALID</i>	for signaling NaN or argument <-1
<i>FE_DIVBYZERO</i>	for -1
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

6.16.4.43 log2

```
half log2 (
    half arg) [friend]
```

Binary logarithm. This function is exact to rounding for all rounding modes.

See also: Documentation for [std::log2](#).

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

logarithm of *arg* to base 2

Exceptions

<i>FE_INVALID</i>	for signaling NaN or negative argument
<i>FE_DIVBYZERO</i>	for 0
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

6.16.4.44 logb

```
half logb (
    half arg) [friend]
```

Extract exponent. **See also:** Documentation for [std::logb](#).

Parameters

<i>arg</i>	number to query
------------	-----------------

Returns

floating-point exponent

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_DIVBYZERO</i>	for 0

6.16.4.45 lrint

```
long lrint (
    half arg) [friend]
```

Nearest integer using half's internal rounding mode. **See also:** Documentation for `std::lrint`.

Parameters

<i>arg</i>	half expression to round
------------	--------------------------

Returns

nearest integer using default rounding mode

Exceptions

<i>FE_INVALID</i>	if value is not representable as `long`
<i>FE_INEXACT</i>	if value had to be rounded

6.16.4.46 lround

```
long lround (
    half arg) [friend]
```

Nearest integer. **See also:** Documentation for `std::lround`.

Parameters

<i>arg</i>	half to round
------------	---------------

Returns

nearest integer, rounded away from zero in half-way cases

Exceptions

<i>FE_INVALID</i>	if value is not representable as `long`
-------------------	---

6.16.4.47 modf

```
half modf (
    half arg,
    half * iptr) [friend]
```

Extract integer and fractional parts. **See also:** Documentation for `std::modf`.

Parameters

<i>arg</i>	number to decompress
<i>iptr</i>	address to store integer part at

Returns

fractional part

Exceptions

<i>FE_INVALID</i>	for signaling NaN
-------------------	-------------------

6.16.4.48 nanh

```
half nanh (  
    const char * arg) [friend]
```

Get NaN value. **See also:** Documentation for `std::nan`.

Parameters

<i>arg</i>	string code
------------	-------------

Returns

quiet NaN

6.16.4.49 nearbyint

```
half nearbyint (  
    half arg) [friend]
```

Nearest integer using half's internal rounding mode. **See also:** Documentation for `std::nearbyint`.

Parameters

<i>arg</i>	half expression to round
------------	--------------------------

Returns

nearest integer using default rounding mode

Exceptions

<i>FE_INVALID</i>	for signaling NaN
-------------------	-------------------

6.16.4.50 nextafter

```
half nextafter (  
    half from,  
    half to) [friend]
```

Next representable value. **See also:** Documentation for `std::nextafter`.

Parameters

<i>from</i>	value to compute next representable value for
<i>to</i>	direction towards which to compute next value

Returns

next representable value after *from* in direction towards *to*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_OVERFLOW</i>	for infinite result from finite argument
<i>FE_UNDERFLOW</i>	for subnormal result

6.16.4.51 nexttoward

```
half nexttoward (
    half from,
    long double to) [friend]
```

Next representable value. **See also:** Documentation for `std::nexttoward`.

Parameters

<i>from</i>	value to compute next representable value for
<i>to</i>	direction towards which to compute next value

Returns

next representable value after *from* in direction towards *to*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_OVERFLOW</i>	for infinite result from finite argument
<i>FE_UNDERFLOW</i>	for subnormal result

6.16.4.52 operator"!="

```
HALF_CONSTEXPR_NOERR bool operator!= (
    half x,
    half y) [friend]
```

Comparison for inequality.

Parameters

<i>x</i>	first operand
<i>y</i>	second operand

Return values

<i>true</i>	if operands not equal
<i>false</i>	else

Exceptions

<i>FE_INVALID</i>	if <i>x</i> or <i>y</i> is NaN
-------------------	--------------------------------

6.16.4.53 operator*

```
half operator* (
    half x,
    half y) [friend]
```

Multiplication. This operation is exact to rounding for all rounding modes.

Parameters

<i>x</i>	left operand
<i>y</i>	right operand

Returns

product of half expressions

Exceptions

<i>FE_INVALID</i>	if multiplying 0 with infinity or if <i>x</i> or <i>y</i> is signaling NaN
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

6.16.4.54 operator+

```
half operator+ (
    half x,
    half y) [friend]
```

Addition. This operation is exact to rounding for all rounding modes.

Parameters

<i>x</i>	left operand
<i>y</i>	right operand

Returns

sum of half expressions

Exceptions

<i>FE_INVALID</i>	if <i>x</i> and <i>y</i> are infinities with different signs or signaling NaNs
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

6.16.4.55 operator- [1/2]

```
HALF_CONSTEXPR half operator- (
    half arg) [friend]
```

Negation.

Parameters

<i>arg</i>	operand
------------	---------

Returns

negated operand

6.16.4.56 operator- [2/2]

```
half operator- (
    half x,
    half y) [friend]
```

Subtraction. This operation is exact to rounding for all rounding modes.

Parameters

<i>x</i>	left operand
<i>y</i>	right operand

Returns

difference of half expressions

Exceptions

<i>FE_INVALID</i>	if <i>x</i> and <i>y</i> are infinities with equal signs or signaling NaNs
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

6.16.4.57 operator/

```
half operator/ (
    half x,
    half y) [friend]
```

Division. This operation is exact to rounding for all rounding modes.

Parameters

<i>x</i>	left operand
<i>y</i>	right operand

Returns

quotient of half expressions

Exceptions

<i>FE_INVALID</i>	if dividing 0s or infinities with each other or if <i>x</i> or <i>y</i> is signaling NaN
<i>FE_DIVBYZERO</i>	if dividing finite value by 0
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

6.16.4.58 operator<

```
HALF_CONSTEXPR_NOERR bool operator< (
    half x,
    half y) [friend]
```

Comparison for less than.

Parameters

<i>x</i>	first operand
<i>y</i>	second operand

Return values

<i>true</i>	if <i>x</i> less than <i>y</i>
<i>false</i>	else

Exceptions

<i>FE_INVALID</i>	if <i>x</i> or <i>y</i> is NaN
-------------------	--------------------------------

6.16.4.59 operator<<

```
template<typename charT, typename traits>
std::basic_ostream< charT, traits > & operator<< (
    std::basic_ostream< charT, traits > & out,
    half arg) [friend]
```

Output operator. This uses the built-in functionality for streaming out floating-point numbers.

Parameters

<i>out</i>	output stream to write into
<i>arg</i>	half expression to write

Returns

reference to output stream

6.16.4.60 operator<=

```
HALF_CONSTEXPR_NOERR bool operator<= (
    half x,
    half y) [friend]
```

Comparison for less equal.

Parameters

<i>x</i>	first operand
<i>y</i>	second operand

Return values

<i>true</i>	if <i>x</i> less equal <i>y</i>
<i>false</i>	else

Exceptions

<i>FE_INVALID</i>	if <i>x</i> or <i>y</i> is NaN
-------------------	--------------------------------

6.16.4.61 operator==

```
HALF_CONSTEXPR_NOERR bool operator== (
    half x,
    half y) [friend]
```

Comparison for equality.

Parameters

<i>x</i>	first operand
<i>y</i>	second operand

Return values

<i>true</i>	if operands equal
<i>false</i>	else

Exceptions

<i>FE_INVALID</i>	if <i>x</i> or <i>y</i> is NaN
-------------------	--------------------------------

6.16.4.62 operator>

```
HALF_CONSTEXPR_NOERR bool operator> (  
    half x,  
    half y) [friend]
```

Comparison for greater than.

Parameters

<i>x</i>	first operand
<i>y</i>	second operand

Return values

<i>true</i>	if <i>x</i> greater than <i>y</i>
<i>false</i>	else

Exceptions

<i>FE_INVALID</i>	if <i>x</i> or <i>y</i> is NaN
-------------------	--------------------------------

6.16.4.63 operator>=

```
HALF_CONSTEXPR_NOERR bool operator>= (  
    half x,  
    half y) [friend]
```

Comparison for greater equal.

Parameters

<i>x</i>	first operand
<i>y</i>	second operand

Return values

<i>true</i>	if <i>x</i> greater equal <i>y</i>
<i>false</i>	else

Exceptions

<i>FE_INVALID</i>	if <i>x</i> or <i>y</i> is NaN
-------------------	--------------------------------

6.16.4.64 operator>>

```
template<typename charT, typename traits>
std::basic_istream< charT, traits > & operator>> (
    std::basic_istream< charT, traits > & in,
    half & arg) [friend]
```

Input operator. This uses the built-in functionality for streaming in floating-point numbers, specifically double precision floating point numbers (unless overridden with `HALF_ARITHMETIC_TYPE`). So the input string is first rounded to double precision using the underlying platform's current floating-point rounding mode before being rounded to half-precision using the library's half-precision rounding mode.

Parameters

<i>in</i>	input stream to read from
<i>arg</i>	half to read into

Returns

reference to input stream

Exceptions

<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding
--	-----------------------

6.16.4.65 pow

```
half pow (
    half x,
    half y) [friend]
```

Power function. This function may be 1 ULP off the correctly rounded exact result for any rounding mode in $\sim 0.00025\%$ of inputs.

See also: Documentation for `std::pow`.

Parameters

<i>x</i>	base
<i>y</i>	exponent

Returns

x raised to *y*

Exceptions

<i>FE_INVALID</i>	if <i>x</i> or <i>y</i> is signaling NaN or if <i>x</i> is finite an negative and <i>y</i> is finite and not integral
<i>FE_DIVBYZERO</i>	if <i>x</i> is 0 and <i>y</i> is negative
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

6.16.4.66 remainder

```
half remainder (
    half x,
    half y) [friend]
```

Remainder of division. **See also:** Documentation for `std::remainder`.

Parameters

<i>x</i>	first operand
<i>y</i>	second operand

Returns

remainder of floating-point division.

Exceptions

<i>FE_INVALID</i>	if <i>x</i> is infinite or <i>y</i> is 0 or if <i>x</i> or <i>y</i> is signaling NaN
-------------------	--

6.16.4.67 remquo

```
half remquo (
    half x,
    half y,
    int * quo) [friend]
```

Remainder of division. **See also:** Documentation for `std::remquo`.

Parameters

<i>x</i>	first operand
<i>y</i>	second operand
<i>quo</i>	address to store some bits of quotient at

Returns

remainder of floating-point division.

Exceptions

<i>FE_INVALID</i>	if <i>x</i> is infinite or <i>y</i> is 0 or if <i>x</i> or <i>y</i> is signaling NaN
-------------------	--

6.16.4.68 rint

```
half rint (
    half arg) [friend]
```

Nearest integer using half's internal rounding mode. **See also:** Documentation for `std::rint`.

Parameters

<i>arg</i>	half expression to round
------------	--------------------------

Returns

nearest integer using default rounding mode

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_INEXACT</i>	if value had to be rounded

6.16.4.69 round

```
half round (
    half arg) [friend]
```

Nearest integer. **See also:** Documentation for `std::round`.

Parameters

<i>arg</i>	half to round
------------	---------------

Returns

nearest integer, rounded away from zero in half-way cases

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_INEXACT</i>	if value had to be rounded

6.16.4.70 rsqrt

```
half rsqrt (
    half arg) [friend]
```

Inverse square root. This function is exact to rounding for all rounding modes and thus generally more accurate than directly computing $1 / \sqrt{arg}$ in half-precision, in addition to also being faster.

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

reciprocal of square root of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN and negative arguments
<i>FE_INEXACT</i>	according to rounding

6.16.4.71 scalbln

```
half scalbln (
    half arg,
    long exp) [friend]
```

Multiply by power of two. This function is exact to rounding for all rounding modes.

See also: Documentation for [std::scalbln](#).

Parameters

<i>arg</i>	number to modify
<i>exp</i>	power of two to multiply with

Returns

arg multiplied by 2 raised to *exp*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

6.16.4.72 signbit

```
HALF_CONSTEXPR bool signbit (
    half arg) [friend]
```

Check sign. **See also:** Documentation for [std::signbit](#).

Parameters

<i>arg</i>	number to check
------------	-----------------

Return values

<i>true</i>	for negative number
<i>false</i>	for positive number

6.16.4.73 sin

```
half sin (
    half arg) [friend]
```

Sine function. This function is exact to rounding for all rounding modes.

See also: Documentation for `std::sin`.

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

sine value of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN or infinity
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

6.16.4.74 sincos

```
void sincos (
    half arg,
    half * sin,
    half * cos) [friend]
```

Compute sine and cosine simultaneously. This returns the same results as `sin()` and `cos()` but is faster than calling each function individually.

This function is exact to rounding for all rounding modes.

Parameters

<i>arg</i>	function argument
<i>sin</i>	variable to take sine of <i>arg</i>
<i>cos</i>	variable to take cosine of <i>arg</i>

Exceptions

<i>FE_INVALID</i>	for signaling NaN or infinity
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

6.16.4.75 sinh

```
half sinh (
    half arg) [friend]
```

Hyperbolic sine. This function is exact to rounding for all rounding modes.

See also: Documentation for [std::sinh](#).

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

hyperbolic sine value of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

6.16.4.76 sqrt

```
half sqrt (
    half arg) [friend]
```

Square root. This function is exact to rounding for all rounding modes.

See also: Documentation for [std::sqrt](#).

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

square root of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN and negative arguments
<i>FE_INEXACT</i>	according to rounding

6.16.4.77 tan

```
half tan (
    half arg) [friend]
```

Tangent function. This function is exact to rounding for all rounding modes.

See also: Documentation for [std::tan](#).

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

tangent value of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN or infinity
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

6.16.4.78 tanh

```
half tanh (
    half arg) [friend]
```

Hyperbolic tangent. This function is exact to rounding for all rounding modes.

See also: Documentation for [std::tanh](#).

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

hyperbolic tangent value of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

6.16.4.79 tgamma

```
half tgamma (
    half arg) [friend]
```

Gamma function. This function may be 1 ULP off the correctly rounded exact result for any rounding mode in <0.25% of inputs.

See also: Documentation for [std::tgamma](#).

Parameters

<i>arg</i>	function argument
------------	-------------------

Returns

gamma function value of *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN, negative infinity or negative integer arguments
<i>FE_DIVBYZERO</i>	for 0
<i>FE_OVERFLOW,...UNDERFLOW,...INEXACT</i>	according to rounding

6.16.4.80 trunc

```
half trunc (
    half arg) [friend]
```

Nearest integer not greater in magnitude than half value. **See also:** Documentation for `std::trunc`.

Parameters

<i>arg</i>	half to round
------------	---------------

Returns

nearest integer not greater in magnitude than *arg*

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_INEXACT</i>	if value had to be rounded

The documentation for this class was generated from the following file:

- GL_Scene/[half.hpp](#)

6.17 half_float::detail::half_caster< T, U, R > Struct Template Reference

```
#include <half.hpp>
```

6.17.1 Detailed Description

```
template<typename T, typename U, std::float_round_style R = (std::float_round_style)(HALF_ROUND_↔
STYLE)>
```

```
struct half_float::detail::half_caster< T, U, R >
```

Helper class for half casts. This class template has to be specialized for all valid cast arguments to define an appropriate static `cast` member function and a corresponding `type` member denoting its return type.

Template Parameters

<i>T</i>	destination type
<i>U</i>	source type
<i>R</i>	rounding mode to use

The documentation for this struct was generated from the following file:

- [GL_Scene/half.hpp](#)

6.18 `half_float::detail::half_caster< half, half, R >` Struct Template Reference

Static Public Member Functions

- static `half cast` (`half` arg)

The documentation for this struct was generated from the following file:

- [GL_Scene/half.hpp](#)

6.19 `half_float::detail::half_caster< half, U, R >` Struct Template Reference

Static Public Member Functions

- static `half cast` (`U` arg)

The documentation for this struct was generated from the following file:

- [GL_Scene/half.hpp](#)

6.20 `half_float::detail::half_caster< T, half, R >` Struct Template Reference

Static Public Member Functions

- static `T cast` (`half` arg)

The documentation for this struct was generated from the following file:

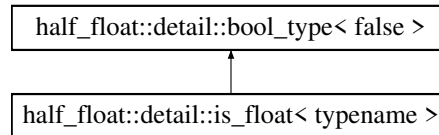
- [GL_Scene/half.hpp](#)

6.21 half_float::detail::is_float< typename > Struct Template Reference

Type traits for floating-point types.

```
#include <half.hpp>
```

Inheritance diagram for half_float::detail::is_float< typename >:



6.21.1 Detailed Description

```
template<typename>
struct half_float::detail::is_float< typename >
```

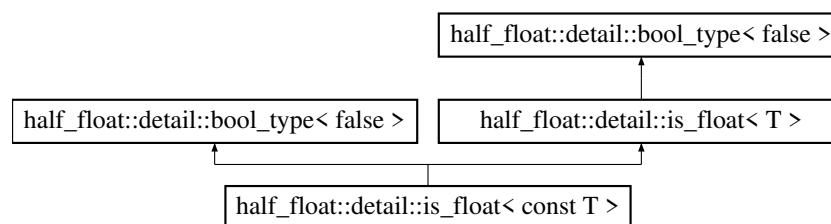
Type traits for floating-point types.

The documentation for this struct was generated from the following file:

- GL_Scene/[half.hpp](#)

6.22 half_float::detail::is_float< const T > Struct Template Reference

Inheritance diagram for half_float::detail::is_float< const T >:

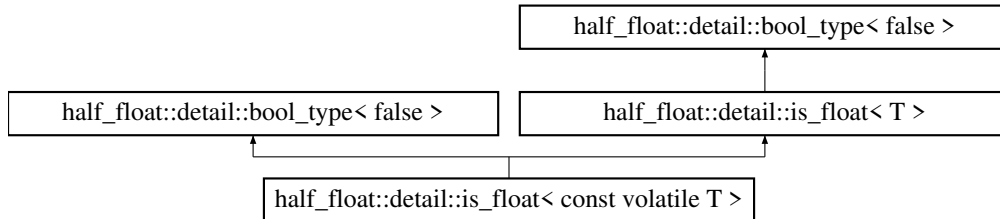


The documentation for this struct was generated from the following file:

- GL_Scene/[half.hpp](#)

6.23 `half_float::detail::is_float< const volatile T >` Struct Template Reference

Inheritance diagram for `half_float::detail::is_float< const volatile T >`:

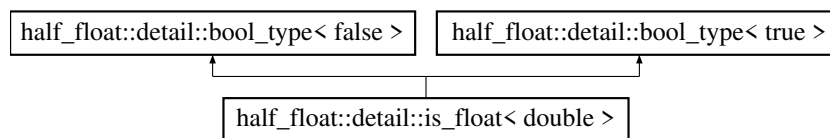


The documentation for this struct was generated from the following file:

- [GL_Scene/half.hpp](#)

6.24 `half_float::detail::is_float< double >` Struct Reference

Inheritance diagram for `half_float::detail::is_float< double >`:

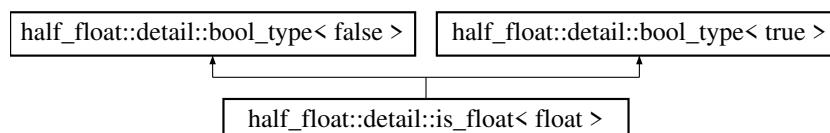


The documentation for this struct was generated from the following file:

- [GL_Scene/half.hpp](#)

6.25 `half_float::detail::is_float< float >` Struct Reference

Inheritance diagram for `half_float::detail::is_float< float >`:

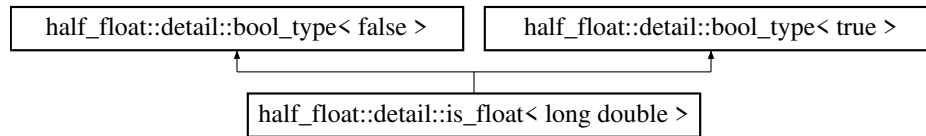


The documentation for this struct was generated from the following file:

- [GL_Scene/half.hpp](#)

6.26 half_float::detail::is_float< long double > Struct Reference

Inheritance diagram for half_float::detail::is_float< long double >:

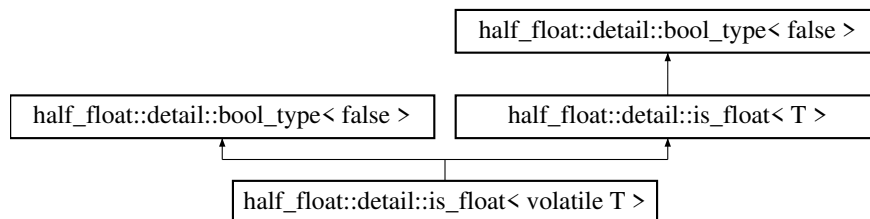


The documentation for this struct was generated from the following file:

- [GL_Scene/half.hpp](#)

6.27 half_float::detail::is_float< volatile T > Struct Template Reference

Inheritance diagram for half_float::detail::is_float< volatile T >:



The documentation for this struct was generated from the following file:

- [GL_Scene/half.hpp](#)

6.28 udit::Light Class Reference

Clase que representa una fuente de luz en la escena.

```
#include <Light.hpp>
```

Public Member Functions

- [Light](#) (const glm::vec3 &pos, const glm::vec3 &col, float ambient, float diffuse)
Constructor de la clase [Light](#).
- void [send_to_shader](#) (GLuint program_id) const
Envía los parámetros de la luz al shader.

Static Public Member Functions

- static std::shared_ptr< [Light](#) > [make_light](#) (const glm::vec3 &pos, const glm::vec3 &col, float ambient, float diffuse)

Crea una luz a partir de los parámetros especificados.

6.28.1 Detailed Description

Clase que representa una fuente de luz en la escena.

La clase [Light](#) es responsable de definir las características básicas de una fuente de luz, tales como su posición, color y las intensidades de la luz ambiental y difusa. Esta clase se utiliza para enviar la información de la luz a los shaders en OpenGL para que los efectos de luz sean aplicados en la escena 3D.

6.28.2 Constructor & Destructor Documentation

6.28.2.1 [Light](#)()

```
Light::Light (
    const glm::vec3 & pos,
    const glm::vec3 & col,
    float ambient,
    float diffuse)
```

Constructor de la clase [Light](#).

Este constructor inicializa los parámetros de la luz con valores específicos para su posición, color y las intensidades de luz ambiental y difusa.

Parameters

<i>pos</i>	Posición de la luz en el espacio 3D.
<i>col</i>	Color de la luz, especificado en formato RGB.
<i>ambient</i>	Intensidad de la luz ambiental.
<i>diffuse</i>	Intensidad de la luz difusa.

6.28.3 Member Function Documentation

6.28.3.1 [make_light](#)()

```
std::shared_ptr< Light > Light::make_light (
    const glm::vec3 & pos,
    const glm::vec3 & col,
    float ambient,
    float diffuse) [static]
```

Crea una luz a partir de los parámetros especificados.

Esta función estática facilita la creación de un objeto [Light](#) compartido (`shared_ptr`) con los valores de posición, color e intensidades de luz ambiental y difusa.

Parameters

<i>pos</i>	Posición de la luz en el espacio 3D.
<i>col</i>	Color de la luz, especificado en formato RGB.
<i>ambient</i>	Intensidad de la luz ambiental.
<i>diffuse</i>	Intensidad de la luz difusa.

Returns

Un `std::shared_ptr<Light>` que apunta a la nueva luz creada.

6.28.3.2 send_to_shader()

```
void Light::send_to_shader (
    GLuint program_id) const
```

Envía los parámetros de la luz al shader.

Esta función toma los parámetros de la luz (posición, color, intensidad) y los envía al shader especificado a través de su programa de OpenGL. Esto permite que la luz sea utilizada en los cálculos de sombreado dentro del pipeline de gráficos.

Parameters

<i>program_id</i>	El identificador del programa de shader de OpenGL.
-------------------	--

The documentation for this class was generated from the following files:

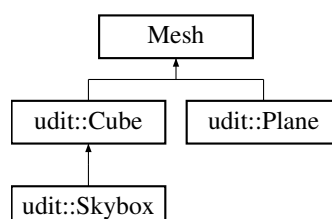
- GL_Scene/Light.hpp
- GL_Scene/Light.cpp

6.29 Mesh Class Reference

Clase que representa una malla 3D.

```
#include <Mesh.hpp>
```

Inheritance diagram for Mesh:



Public Member Functions

- **Mesh ()**
Constructor por defecto.
- **Mesh (std::string &path)**
Constructor que carga una malla desde un archivo.
- virtual **~Mesh ()**
Destructor de la clase.
- virtual void **translate** (glm::vec3 translation)
Realiza una traslación de la malla.
- virtual void **rotate** (glm::vec3 rotation, float angle)
Rota la malla.
- virtual void **scale** (glm::vec3 scale)
Escala la malla.
- virtual void **update** ()
Actualiza la malla.
- virtual void **render** (glm::mat4 view_matrix)
Renderiza la malla.
- virtual void **resize** (glm::mat4 projection_matrix)
Ajusta la matriz de proyección.
- virtual void **set_shader** (std::shared_ptr< **udit::Shader** > shader)
Asocia un shader a la malla.
- GLuint **get_shader_program_id** () const
Obtiene el ID del programa del shader asociado.
- std::vector< GLint > **get_shader_matrix_ids** ()
Obtiene los IDs de las matrices del shader asociadas a la malla.
- glm::mat4 **get_model_view_matrix** () const
Obtiene la matriz de transformación del modelo.
- void **set_model_view_matrix** (glm::mat4 matrix)
Establece la matriz de transformación del modelo.
- void **set_mesh_type** (MeshType type)
Establece el tipo de malla.

Static Public Member Functions

- static std::shared_ptr< **Mesh** > **make_mesh** (MeshType type, const std::string &path="")
Crea una malla de un tipo específico.

Protected Member Functions

- void **create_mesh** (std::string mesh_name="")
Crea los VBOs y el VAO necesarios para la malla.

Protected Attributes

- std::vector< glm::vec3 > **coordinates**
Vectores que almacenan las coordenadas de los vértices, colores, normales, índices y coordenadas de textura.
- std::vector< glm::vec3 > **colors**
- std::vector< glm::vec3 > **normals**
- std::vector< GLuint > **indices**
- std::vector< glm::vec2 > **texture_uvs**
- GLsizei **number_of_vertices**
Número total de vértices de la malla.

6.29.1 Detailed Description

Clase que representa una malla 3D.

La clase [Mesh](#) es la base para representar mallas 3D en OpenGL. Contiene todos los atributos y funciones necesarias para cargar, gestionar y renderizar mallas con vértices, normales, colores, coordenadas de textura y los índices que definen la topología de la malla. Esta clase también incluye funciones para transformar la malla (traslación, rotación, escala) y para actualizar y renderizar la malla en la escena.

6.29.2 Constructor & Destructor Documentation

6.29.2.1 Mesh()

```
udit::Mesh::Mesh (  
    std::string & path)
```

Constructor que carga una malla desde un archivo.

Este constructor carga los datos de la malla (coordenadas, normales, colores, etc.) desde un archivo y los almacena en los atributos correspondientes.

Parameters

<i>path</i>	Ruta al archivo que contiene la malla.
-------------	--

6.29.2.2 ~Mesh()

```
udit::Mesh::~Mesh () [virtual]
```

Destructor de la clase.

El destructor limpia los recursos de OpenGL, como los buffers y el VAO.

6.29.3 Member Function Documentation

6.29.3.1 create_mesh()

```
void udit::Mesh::create_mesh (  
    std::string mesh_name = "") [protected]
```

Crea los VBOs y el VAO necesarios para la malla.

Parameters

<i>mesh_name</i>	Nombre de la malla a crear.
------------------	-----------------------------

6.29.3.2 get_model_view_matrix()

```
glm::mat4 udit::Mesh::get_model_view_matrix () const [inline]
```

Obtiene la matriz de transformación del modelo.

Returns

La matriz de transformación del modelo.

6.29.3.3 get_shader_matrix_ids()

```
std::vector< GLint > udit::Mesh::get_shader_matrix_ids ()
```

Obtiene los IDs de las matrices del shader asociadas a la malla.

Devuelve los IDs de las matrices necesarias para renderizar la malla en el shader.

Returns

Un vector con los IDs de las matrices.

6.29.3.4 get_shader_program_id()

```
GLuint udit::Mesh::get_shader_program_id () const
```

Obtiene el ID del programa del shader asociado.

Returns

El ID del programa de shader asociado a la malla.

6.29.3.5 make_mesh()

```
std::shared_ptr< Mesh > udit::Mesh::make_mesh (  
    MeshType type,  
    const std::string & path = "") [static]
```

Crea una malla de un tipo específico.

Este método estático permite crear una malla de un tipo específico, como terreno, malla básica, o malla cargada desde un archivo.

Parameters

<i>type</i>	Tipo de malla a crear.
<i>path</i>	Ruta al archivo de la malla (solo relevante si el tipo es MESH).

Returns

Un puntero compartido a la malla creada.

6.29.3.6 render()

```
void udit::Mesh::render (  
    glm::mat4 view_matrix) [virtual]
```

Renderiza la malla.

Función de renderizado de la malla en el bucle principal.

Utiliza el shader asociado y la matriz de vista para renderizar la malla.

Parameters

<i>view_matrix</i>	Matriz de vista.
--------------------	------------------

6.29.3.7 resize()

```
void udit::Mesh::resize (
    glm::mat4 projection_matrix) [virtual]
```

Ajusta la matriz de proyección.

Establece la matriz de proyección en el shader para la correcta visualización.

Parameters

<i>projection_matrix</i>	Matriz de proyección.
--------------------------	-----------------------

6.29.3.8 rotate()

```
void udit::Mesh::rotate (
    glm::vec3 rotation,
    float angle) [virtual]
```

Rota la malla.

Aplica una rotación a la matriz de transformación de la malla.

Parameters

<i>rotation</i>	Eje de rotación.
<i>angle</i>	Ángulo de rotación en grados.

6.29.3.9 scale()

```
void udit::Mesh::scale (
    glm::vec3 scale) [virtual]
```

Escala la malla.

Aplica una escala a la matriz de transformación de la malla.

Parameters

<i>scale</i>	Factor de escala.
--------------	-------------------

6.29.3.10 set_mesh_type()

```
void udit::Mesh::set_mesh_type (
    MeshType type) [inline]
```

Establece el tipo de malla.

Parameters

<i>type</i>	Tipo de malla.
-------------	----------------

6.29.3.11 set_model_view_matrix()

```
void udit::Mesh::set_model_view_matrix (
    glm::mat4 matrix) [inline]
```

Establece la matriz de transformación del modelo.

Parameters

<i>matrix</i>	Nueva matriz de transformación del modelo.
---------------	--

6.29.3.12 set_shader()

```
void udit::Mesh::set_shader (
    std::shared_ptr< udit::Shader > shader) [virtual]
```

Asocia un shader a la malla.

Permite asociar un shader para ser usado al renderizar la malla.

Parameters

<i>shader</i>	Puntero al shader a asociar.
---------------	------------------------------

6.29.3.13 translate()

```
void udit::Mesh::translate (
    glm::vec3 translation) [virtual]
```

Realiza una traslación de la malla.

Aplica una traslación a la matriz de transformación de la malla.

Parameters

<i>translation</i>	Vector de traslación.
--------------------	-----------------------

6.29.3.14 update()

```
void udit::Mesh::update () [virtual]
```

Actualiza la malla.

Función de actualización de la malla en el bucle principal.

Esta función puede ser utilizada para actualizar los datos de la malla, si es necesario.

The documentation for this class was generated from the following files:

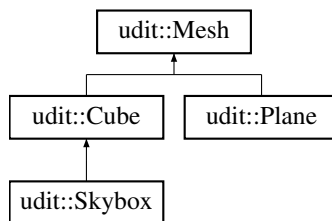
- GL_Scene/Mesh.hpp
- GL_Scene/Mesh.cpp

6.30 udit::Mesh Class Reference

Clase que representa una malla 3D.

```
#include <Mesh.hpp>
```

Inheritance diagram for udit::Mesh:



Public Member Functions

- **Mesh ()**
Constructor por defecto.
- **Mesh (std::string &path)**
Constructor que carga una malla desde un archivo.
- virtual **~Mesh ()**
Destructor de la clase.
- virtual void **translate** (glm::vec3 translation)
Realiza una traslación de la malla.
- virtual void **rotate** (glm::vec3 rotation, float angle)
Rota la malla.
- virtual void **scale** (glm::vec3 scale)
Escala la malla.
- virtual void **update ()**
Actualiza la malla.
- virtual void **render** (glm::mat4 view_matrix)
Renderiza la malla.
- virtual void **resize** (glm::mat4 projection_matrix)

- *Ajusta la matriz de proyección.*
virtual void [set_shader](#) (std::shared_ptr< [udit::Shader](#) > shader)
- *Asocia un shader a la malla.*
GLuint [get_shader_program_id](#) () const
- *Obtiene el ID del programa del shader asociado.*
std::vector< GLint > [get_shader_matrix_ids](#) ()
- *Obtiene los IDs de las matrices del shader asociadas a la malla.*
glm::mat4 [get_model_view_matrix](#) () const
- *Obtiene la matriz de transformación del modelo.*
void [set_model_view_matrix](#) (glm::mat4 matrix)
- *Establece la matriz de transformación del modelo.*
void [set_mesh_type](#) (MeshType type)
- *Establece el tipo de malla.*

Static Public Member Functions

- static std::shared_ptr< [Mesh](#) > [make_mesh](#) (MeshType type, const std::string &path="")
Crea una malla de un tipo específico.

Protected Member Functions

- void [create_mesh](#) (std::string mesh_name="")
Crea los VBOs y el VAO necesarios para la malla.

Protected Attributes

- std::vector< glm::vec3 > **coordinates**
Vectores que almacenan las coordenadas de los vértices, colores, normales, índices y coordenadas de textura.
- std::vector< glm::vec3 > **colors**
- std::vector< glm::vec3 > **normals**
- std::vector< GLuint > **indices**
- std::vector< glm::vec2 > **texture_uvs**
- GLsizei **number_of_vertices**
Número total de vértices de la malla.

6.30.1 Detailed Description

Clase que representa una malla 3D.

La clase [Mesh](#) es la base para representar mallas 3D en OpenGL. Contiene todos los atributos y funciones necesarias para cargar, gestionar y renderizar mallas con vértices, normales, colores, coordenadas de textura y los índices que definen la topología de la malla. Esta clase también incluye funciones para transformar la malla (traslación, rotación, escala) y para actualizar y renderizar la malla en la escena.

6.30.2 Constructor & Destructor Documentation

6.30.2.1 Mesh()

```
udit::Mesh::Mesh (
    std::string & path)
```

Constructor que carga una malla desde un archivo.

Este constructor carga los datos de la malla (coordenadas, normales, colores, etc.) desde un archivo y los almacena en los atributos correspondientes.

Parameters

<i>path</i>	Ruta al archivo que contiene la malla.
-------------	--

6.30.2.2 ~Mesh()

```
udit::Mesh::~Mesh () [virtual]
```

Destructor de la clase.

El destructor limpia los recursos de OpenGL, como los buffers y el VAO.

6.30.3 Member Function Documentation**6.30.3.1 create_mesh()**

```
void udit::Mesh::create_mesh (  
    std::string mesh_name = "") [protected]
```

Crea los VBOs y el VAO necesarios para la malla.

Parameters

<i>mesh_name</i>	Nombre de la malla a crear.
------------------	-----------------------------

6.30.3.2 get_model_view_matrix()

```
glm::mat4 udit::Mesh::get_model_view_matrix () const [inline]
```

Obtiene la matriz de transformación del modelo.

Returns

La matriz de transformación del modelo.

6.30.3.3 get_shader_matrix_ids()

```
std::vector< GLint > udit::Mesh::get_shader_matrix_ids ()
```

Obtiene los IDs de las matrices del shader asociadas a la malla.

Devuelve los IDs de las matrices necesarias para renderizar la malla en el shader.

Returns

Un vector con los IDs de las matrices.

6.30.3.4 get_shader_program_id()

```
GLuint udit::Mesh::get_shader_program_id () const
```

Obtiene el ID del programa del shader asociado.

Returns

El ID del programa de shader asociado a la malla.

6.30.3.5 make_mesh()

```
std::shared_ptr< Mesh > udit::Mesh::make_mesh (
    MeshType type,
    const std::string & path = "") [static]
```

Crea una malla de un tipo específico.

Este método estático permite crear una malla de un tipo específico, como terreno, malla básica, o malla cargada desde un archivo.

Parameters

<i>type</i>	Tipo de malla a crear.
<i>path</i>	Ruta al archivo de la malla (solo relevante si el tipo es MESH).

Returns

Un puntero compartido a la malla creada.

6.30.3.6 render()

```
void udit::Mesh::render (
    glm::mat4 view_matrix) [virtual]
```

Renderiza la malla.

Función de renderizado de la malla en el bucle principal.

Utiliza el shader asociado y la matriz de vista para renderizar la malla.

Parameters

<i>view_matrix</i>	Matriz de vista.
--------------------	------------------

6.30.3.7 resize()

```
void udit::Mesh::resize (
    glm::mat4 projection_matrix) [virtual]
```

Ajusta la matriz de proyección.

Establece la matriz de proyección en el shader para la correcta visualización.

Parameters

<i>projection_matrix</i>	Matriz de proyección.
--------------------------	-----------------------

6.30.3.8 rotate()

```
void udit::Mesh::rotate (  
    glm::vec3 rotation,  
    float angle) [virtual]
```

Rota la malla.

Aplica una rotación a la matriz de transformación de la malla.

Parameters

<i>rotation</i>	Eje de rotación.
<i>angle</i>	Ángulo de rotación en grados.

6.30.3.9 scale()

```
void udit::Mesh::scale (  
    glm::vec3 scale) [virtual]
```

Escala la malla.

Aplica una escala a la matriz de transformación de la malla.

Parameters

<i>scale</i>	Factor de escala.
--------------	-------------------

6.30.3.10 set_mesh_type()

```
void udit::Mesh::set_mesh_type (  
    MeshType type) [inline]
```

Establece el tipo de malla.

Parameters

<i>type</i>	Tipo de malla.
-------------	----------------

6.30.3.11 set_model_view_matrix()

```
void udit::Mesh::set_model_view_matrix (  
    glm::mat4 matrix) [inline]
```

Establece la matriz de transformación del modelo.

Parameters

<i>matrix</i>	Nueva matriz de transformación del modelo.
---------------	--

6.30.3.12 set_shader()

```
void udit::Mesh::set_shader (  
    std::shared_ptr< udit::Shader > shader) [virtual]
```

Asocia un shader a la malla.

Permite asociar un shader para ser usado al renderizar la malla.

Parameters

<i>shader</i>	Puntero al shader a asociar.
---------------	------------------------------

6.30.3.13 translate()

```
void udit::Mesh::translate (  
    glm::vec3 translation) [virtual]
```

Realiza una traslación de la malla.

Aplica una traslación a la matriz de transformación de la malla.

Parameters

<i>translation</i>	Vector de traslación.
--------------------	-----------------------

6.30.3.14 update()

```
void udit::Mesh::update () [virtual]
```

Actualiza la malla.

Función de actualización de la malla en el bucle principal.

Esta función puede ser utilizada para actualizar los datos de la malla, si es necesario.

The documentation for this class was generated from the following files:

- GL_Scene/Mesh.hpp
- GL_Scene/Mesh.cpp

6.31 std::numeric_limits< half_float::half > Class Reference

```
#include <half.hpp>
```


Static Public Member Functions

- static HALF_CONSTEXPR [half_float::half](#) **min** () HALF_NOTHROW
Smallest positive normal value.
- static HALF_CONSTEXPR [half_float::half](#) **lowest** () HALF_NOTHROW
Smallest finite value.
- static HALF_CONSTEXPR [half_float::half](#) **max** () HALF_NOTHROW
Largest finite value.
- static HALF_CONSTEXPR [half_float::half](#) **epsilon** () HALF_NOTHROW
Difference between 1 and next representable value.
- static HALF_CONSTEXPR [half_float::half](#) **round_error** () HALF_NOTHROW
Maximum rounding error in ULP (units in the last place).
- static HALF_CONSTEXPR [half_float::half](#) **infinity** () HALF_NOTHROW
Positive infinity.
- static HALF_CONSTEXPR [half_float::half](#) **quiet_NaN** () HALF_NOTHROW
Quiet NaN.
- static HALF_CONSTEXPR [half_float::half](#) **signaling_NaN** () HALF_NOTHROW
Signaling NaN.
- static HALF_CONSTEXPR [half_float::half](#) **denorm_min** () HALF_NOTHROW
Smallest positive subnormal value.

Static Public Attributes

- static HALF_CONSTEXPR_CONST bool **is_specialized** = true
Is template specialization.
- static HALF_CONSTEXPR_CONST bool **is_signed** = true
Supports signed values.
- static HALF_CONSTEXPR_CONST bool **is_integer** = false
Is not an integer type.
- static HALF_CONSTEXPR_CONST bool **is_exact** = false
Is not exact.
- static HALF_CONSTEXPR_CONST bool **is_modulo** = false
Doesn't provide modulo arithmetic.
- static HALF_CONSTEXPR_CONST bool **is_bounded** = true
Has a finite set of values.
- static HALF_CONSTEXPR_CONST bool **is_iec559** = true
IEEE conformant.
- static HALF_CONSTEXPR_CONST bool **has_infinity** = true
Supports infinity.
- static HALF_CONSTEXPR_CONST bool **has_quiet_NaN** = true
Supports quiet NaNs.
- static HALF_CONSTEXPR_CONST bool **has_signaling_NaN** = true
Supports signaling NaNs.
- static HALF_CONSTEXPR_CONST float_denorm_style **has_denorm** = denorm_present
Supports subnormal values.
- static HALF_CONSTEXPR_CONST bool **has_denorm_loss** = false
Supports no denormalization detection.
- static HALF_CONSTEXPR_CONST bool **traps** = false
Traps only if HALF_ERRHANDLING_THROW... is acitvated.
- static HALF_CONSTEXPR_CONST bool **tinyness_before** = false

- Does not support no pre-rounding underflow detection.*
- static HALF_CONSTEXPR_CONST float_round_style **round_style** = half_float::half::round_style
- Rounding mode.*
- static HALF_CONSTEXPR_CONST int **digits** = 11
- Significant digits.*
- static HALF_CONSTEXPR_CONST int **digits10** = 3
- Significant decimal digits.*
- static HALF_CONSTEXPR_CONST int **max_digits10** = 5
- Required decimal digits to represent all possible values.*
- static HALF_CONSTEXPR_CONST int **radix** = 2
- Number base.*
- static HALF_CONSTEXPR_CONST int **min_exponent** = -13
- One more than smallest exponent.*
- static HALF_CONSTEXPR_CONST int **min_exponent10** = -4
- Smallest normalized representable power of 10.*
- static HALF_CONSTEXPR_CONST int **max_exponent** = 16
- One more than largest exponent.*
- static HALF_CONSTEXPR_CONST int **max_exponent10** = 4
- Largest finitely representable power of 10.*

6.31.1 Detailed Description

Numeric limits for half-precision floats. **See also:** Documentation for [std::numeric_limits](#)

The documentation for this class was generated from the following file:

- [GL_Scene/half.hpp](#)

6.32 udit::Window::OpenGL_Context_Settings Struct Reference

Public Attributes

- unsigned **version_major** = 3
- unsigned **version_minor** = 3
- bool **core_profile** = true
- unsigned **depth_buffer_size** = 24
- unsigned **stencil_buffer_size** = 0
- bool **enable_vsync** = true

The documentation for this struct was generated from the following file:

- [GL_Scene/Window.hpp](#)

6.33 Window::OpenGL_Context_Settings Struct Reference

Public Attributes

- unsigned **version_major** = 3
- unsigned **version_minor** = 3
- bool **core_profile** = true
- unsigned **depth_buffer_size** = 24
- unsigned **stencil_buffer_size** = 0
- bool **enable_vsync** = true

The documentation for this struct was generated from the following file:

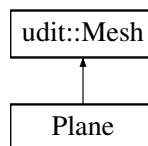
- GL_Scene/Window.hpp

6.34 Plane Class Reference

Clase que representa un plano 3D.

```
#include <Plane.hpp>
```

Inheritance diagram for Plane:



Public Member Functions

- [Plane](#) ()
Constructor por defecto.
- [Plane](#) (float size)
Constructor que define el tamaño del plano.
- [Plane](#) (float width, float height, unsigned columns, unsigned rows)
Constructor que define el tamaño y la resolución del plano.

Public Member Functions inherited from [udit::Mesh](#)

- **Mesh** ()
Constructor por defecto.
- **Mesh** (std::string &path)
Constructor que carga una malla desde un archivo.
- virtual **~Mesh** ()
Destructor de la clase.
- virtual void **translate** (glm::vec3 translation)
Realiza una traslación de la malla.
- virtual void **rotate** (glm::vec3 rotation, float angle)
Rota la malla.
- virtual void **scale** (glm::vec3 scale)
Escala la malla.
- virtual void **update** ()
Actualiza la malla.
- virtual void **render** (glm::mat4 view_matrix)
Renderiza la malla.
- virtual void **resize** (glm::mat4 projection_matrix)
Ajusta la matriz de proyección.
- virtual void **set_shader** (std::shared_ptr< [udit::Shader](#) > shader)
Asocia un shader a la malla.
- GLuint **get_shader_program_id** () const
Obtiene el ID del programa del shader asociado.
- std::vector< GLint > **get_shader_matrix_ids** ()
Obtiene los IDs de las matrices del shader asociadas a la malla.
- glm::mat4 **get_model_view_matrix** () const
Obtiene la matriz de transformación del modelo.
- void **set_model_view_matrix** (glm::mat4 matrix)
Establece la matriz de transformación del modelo.
- void **set_mesh_type** (MeshType type)
Establece el tipo de malla.

Additional Inherited Members

Static Public Member Functions inherited from [udit::Mesh](#)

- static std::shared_ptr< [Mesh](#) > **make_mesh** (MeshType type, const std::string &path="")
Crea una malla de un tipo específico.

Protected Member Functions inherited from [udit::Mesh](#)

- void **create_mesh** (std::string mesh_name="")
Crea los VBOs y el VAO necesarios para la malla.

Protected Attributes inherited from [udit::Mesh](#)

- `std::vector< glm::vec3 >` **coordinates**
Vectores que almacenan las coordenadas de los vértices, colores, normales, índices y coordenadas de textura.
- `std::vector< glm::vec3 >` **colors**
- `std::vector< glm::vec3 >` **normals**
- `std::vector< GLuint >` **indices**
- `std::vector< glm::vec2 >` **texture_uvs**
- `GLsizei` **number_of_vertices**
Número total de vértices de la malla.

6.34.1 Detailed Description

Clase que representa un plano 3D.

La clase [Plane](#) hereda de [Mesh](#) y está diseñada para representar un plano 3D en OpenGL. El plano se define por su ancho, altura, y la cantidad de columnas y filas que tiene. Esta clase permite crear un plano con diferentes configuraciones, ya sea con un tamaño específico o con una distribución de vértices más compleja. El plano es útil para representar superficies planas, como terrenos o fondos.

6.34.2 Constructor & Destructor Documentation

6.34.2.1 [Plane\(\)](#) [1/3]

```
udit::Plane::Plane ()
```

Constructor por defecto.

Crea un plano con dimensiones predeterminadas.

6.34.2.2 [Plane\(\)](#) [2/3]

```
udit::Plane::Plane (
    float size)
```

Constructor que define el tamaño del plano.

Crea un plano cuadrado con el tamaño especificado.

Parameters

<i>size</i>	Tamaño del plano en ambas dimensiones (ancho y alto).
-------------	---

6.34.2.3 [Plane\(\)](#) [3/3]

```
udit::Plane::Plane (
    float width,
    float height,
    unsigned columns,
    unsigned rows)
```

Constructor que define el tamaño y la resolución del plano.

Crea un plano con el tamaño y la cantidad de columnas y filas especificados.

Parameters

<i>width</i>	Ancho del plano.
<i>height</i>	Alto del plano.
<i>columns</i>	Número de columnas del plano (resolución horizontal).
<i>rows</i>	Número de filas del plano (resolución vertical).

The documentation for this class was generated from the following files:

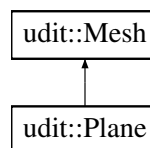
- GL_Scene/Plane.hpp
- GL_Scene/Plane.cpp

6.35 udit::Plane Class Reference

Clase que representa un plano 3D.

```
#include <Plane.hpp>
```

Inheritance diagram for udit::Plane:



Public Member Functions

- [Plane](#) ()
Constructor por defecto.
- [Plane](#) (float size)
Constructor que define el tamaño del plano.
- [Plane](#) (float width, float height, unsigned columns, unsigned rows)
Constructor que define el tamaño y la resolución del plano.

Public Member Functions inherited from [udit::Mesh](#)

- [Mesh](#) ()
Constructor por defecto.
- [Mesh](#) (std::string &path)
Constructor que carga una malla desde un archivo.
- virtual [~Mesh](#) ()
Destructor de la clase.
- virtual void [translate](#) (glm::vec3 translation)
Realiza una traslación de la malla.
- virtual void [rotate](#) (glm::vec3 rotation, float angle)
Rota la malla.

- virtual void [scale](#) (glm::vec3 scale)
Escala la malla.
- virtual void [update](#) ()
Actualiza la malla.
- virtual void [render](#) (glm::mat4 view_matrix)
Renderiza la malla.
- virtual void [resize](#) (glm::mat4 projection_matrix)
Ajusta la matriz de proyección.
- virtual void [set_shader](#) (std::shared_ptr< [udit::Shader](#) > shader)
Asocia un shader a la malla.
- GLuint [get_shader_program_id](#) () const
Obtiene el ID del programa del shader asociado.
- std::vector< GLint > [get_shader_matrix_ids](#) ()
Obtiene los IDs de las matrices del shader asociadas a la malla.
- glm::mat4 [get_model_view_matrix](#) () const
Obtiene la matriz de transformación del modelo.
- void [set_model_view_matrix](#) (glm::mat4 matrix)
Establece la matriz de transformación del modelo.
- void [set_mesh_type](#) (MeshType type)
Establece el tipo de malla.

Additional Inherited Members

Static Public Member Functions inherited from [udit::Mesh](#)

- static std::shared_ptr< [Mesh](#) > [make_mesh](#) (MeshType type, const std::string &path="")
Crea una malla de un tipo específico.

Protected Member Functions inherited from [udit::Mesh](#)

- void [create_mesh](#) (std::string mesh_name="")
Crea los VBOs y el VAO necesarios para la malla.

Protected Attributes inherited from [udit::Mesh](#)

- std::vector< glm::vec3 > **coordinates**
Vectores que almacenan las coordenadas de los vértices, colores, normales, índices y coordenadas de textura.
- std::vector< glm::vec3 > **colors**
- std::vector< glm::vec3 > **normals**
- std::vector< GLuint > **indices**
- std::vector< glm::vec2 > **texture_uvs**
- GLsizei **number_of_vertices**
Número total de vértices de la malla.

6.35.1 Detailed Description

Clase que representa un plano 3D.

La clase `Plane` hereda de `Mesh` y está diseñada para representar un plano 3D en OpenGL. El plano se define por su ancho, altura, y la cantidad de columnas y filas que tiene. Esta clase permite crear un plano con diferentes configuraciones, ya sea con un tamaño específico o con una distribución de vértices más compleja. El plano es útil para representar superficies planas, como terrenos o fondos.

6.35.2 Constructor & Destructor Documentation

6.35.2.1 `Plane()` [1/3]

```
udit::Plane::Plane ()
```

Constructor por defecto.

Crea un plano con dimensiones predeterminadas.

6.35.2.2 `Plane()` [2/3]

```
udit::Plane::Plane (
    float size)
```

Constructor que define el tamaño del plano.

Crea un plano cuadrado con el tamaño especificado.

Parameters

<i>size</i>	Tamaño del plano en ambas dimensiones (ancho y alto).
-------------	---

6.35.2.3 `Plane()` [3/3]

```
udit::Plane::Plane (
    float width,
    float height,
    unsigned columns,
    unsigned rows)
```

Constructor que define el tamaño y la resolución del plano.

Crea un plano con el tamaño y la cantidad de columnas y filas especificados.

Parameters

<i>width</i>	Ancho del plano.
<i>height</i>	Alto del plano.
<i>columns</i>	Número de columnas del plano (resolución horizontal).
<i>rows</i>	Número de filas del plano (resolución vertical).

The documentation for this class was generated from the following files:

- `GL_Scene/Plane.hpp`
- `GL_Scene/Plane.cpp`

6.36 Scene Class Reference

Representa una escena 3D con un skybox, terreno, luz y otros elementos.

```
#include <Scene.hpp>
```

Public Member Functions

- [Scene](#) (unsigned width, unsigned height)
Constructor de la escena.
- void [update](#) ()
Actualiza la escena.
- void [render](#) ()
Renderiza la escena.
- void [resize](#) (unsigned width, unsigned height)
Redimensiona la escena.
- void [set_view_matrix](#) (const glm::mat4 &view)
Establece la matriz de vista para la cámara.
- void [set_projection_matrix](#) (const glm::mat4 &projection)
Establece la matriz de proyección para la cámara.
- void [set_lights](#) (GLuint shader_program_id)
Establece las luces en el shader.

6.36.1 Detailed Description

Representa una escena 3D con un skybox, terreno, luz y otros elementos.

La clase [Scene](#) es responsable de gestionar la representación de una escena 3D, incluyendo los objetos gráficos principales y la iluminación. Los métodos permiten actualizar la escena, renderizarla y ajustar su tamaño.

6.36.2 Constructor & Destructor Documentation

6.36.2.1 Scene()

```
udit::Scene::Scene (
    unsigned width,
    unsigned height)
```

Constructor de la escena.

Constructor.

Inicializa una nueva escena con el ancho y alto especificados.

Parameters

<i>width</i>	Ancho de la ventana de renderizado.
<i>height</i>	Alto de la ventana de renderizado.

Inicializa una escena

Parameters

<i>width</i>	Ancho de la escena
<i>height</i>	Alto de la escena

6.36.3 Member Function Documentation

6.36.3.1 render()

```
void udit::Scene::render ()
```

Renderiza la escena.

Renderiza los elementos de la escena.

Dibuja todos los elementos de la escena (skybox, terreno, objetos, luz) en la ventana de renderizado. Este método debe ser llamado en cada ciclo de renderizado.

6.36.3.2 resize()

```
void udit::Scene::resize (
    unsigned width,
    unsigned height)
```

Redimensiona la escena.

Ajusta la escena al nuevo tamaño de la ventana.

Parameters

<i>width</i>	Nuevo ancho de la ventana.
<i>height</i>	Nuevo alto de la ventana.

6.36.3.3 set_lights()

```
void udit::Scene::set_lights (
    GLuint shader_program_id)
```

Establece las luces en el shader.

Configura las luces de la escena dentro del shader, enviando los parámetros necesarios al programa de sombreado.

Parameters

<i>shader_program_id</i>	Identificador del programa de sombreado (shader).
--------------------------	---

6.36.3.4 set_projection_matrix()

```
void udit::Scene::set_projection_matrix (
    const glm::mat4 & projection)
```

Establece la matriz de proyección para la cámara.

Establece la matriz de proyección que será usada para renderizar la escena.

Parameters

<i>projection</i>	Matriz de proyección.
-------------------	-----------------------

6.36.3.5 set_view_matrix()

```
void udit::Scene::set_view_matrix (
    const glm::mat4 & view)
```

Establece la matriz de vista para la cámara.

Establece la matriz de vista que será usada para renderizar la escena.

Parameters

<i>view</i>	Matriz de vista.
-------------	------------------

6.36.3.6 update()

```
void udit::Scene::update ()
```

Actualiza la escena.

Actualiza ciertos valores dentro del bucle principal.

Llama a las funciones necesarias para actualizar los objetos en la escena. Este método debe ser llamado cada vez que se desea actualizar el estado de la escena.

The documentation for this class was generated from the following files:

- GL_Scene/[Scene.hpp](#)
- GL_Scene/Scene.cpp

6.37 udit::Scene Class Reference

Representa una escena 3D con un skybox, terreno, luz y otros elementos.

```
#include <Scene.hpp>
```

Public Member Functions

- [Scene](#) (unsigned width, unsigned height)
Constructor de la escena.
- void [update](#) ()
Actualiza la escena.
- void [render](#) ()
Renderiza la escena.
- void [resize](#) (unsigned width, unsigned height)
Redimensiona la escena.
- void [set_view_matrix](#) (const glm::mat4 &view)
Establece la matriz de vista para la cámara.
- void [set_projection_matrix](#) (const glm::mat4 &projection)
Establece la matriz de proyección para la cámara.
- void [set_lights](#) (GLuint shader_program_id)
Establece las luces en el shader.

6.37.1 Detailed Description

Representa una escena 3D con un skybox, terreno, luz y otros elementos.

La clase [Scene](#) es responsable de gestionar la representación de una escena 3D, incluyendo los objetos gráficos principales y la iluminación. Los métodos permiten actualizar la escena, renderizarla y ajustar su tamaño.

6.37.2 Constructor & Destructor Documentation

6.37.2.1 Scene()

```
udit::Scene::Scene (  
    unsigned width,  
    unsigned height)
```

Constructor de la escena.

Constructor.

Inicializa una nueva escena con el ancho y alto especificados.

Parameters

<i>width</i>	Ancho de la ventana de renderizado.
<i>height</i>	Alto de la ventana de renderizado.

Inicializa una escena

Parameters

<i>width</i>	Ancho de la escena
<i>height</i>	Alto de la escena

6.37.3 Member Function Documentation

6.37.3.1 render()

```
void udit::Scene::render ()
```

Renderiza la escena.

Renderiza los elementos de la escena.

Dibuja todos los elementos de la escena (skybox, terreno, objetos, luz) en la ventana de renderizado. Este método debe ser llamado en cada ciclo de renderizado.

6.37.3.2 resize()

```
void udit::Scene::resize (  
    unsigned width,  
    unsigned height)
```

Redimensiona la escena.

Ajusta la escena al nuevo tamaño de la ventana.

Parameters

<i>width</i>	Nuevo ancho de la ventana.
<i>height</i>	Nuevo alto de la ventana.

6.37.3.3 set_lights()

```
void udit::Scene::set_lights (
    GLuint shader_program_id)
```

Establece las luces en el shader.

Configura las luces de la escena dentro del shader, enviando los parámetros necesarios al programa de sombreado.

Parameters

<i>shader_program_id</i>	Identificador del programa de sombreado (shader).
--------------------------	---

6.37.3.4 set_projection_matrix()

```
void udit::Scene::set_projection_matrix (
    const glm::mat4 & projection)
```

Establece la matriz de proyección para la cámara.

Establece la matriz de proyección que será usada para renderizar la escena.

Parameters

<i>projection</i>	Matriz de proyección.
-------------------	-----------------------

6.37.3.5 set_view_matrix()

```
void udit::Scene::set_view_matrix (
    const glm::mat4 & view)
```

Establece la matriz de vista para la cámara.

Establece la matriz de vista que será usada para renderizar la escena.

Parameters

<i>view</i>	Matriz de vista.
-------------	------------------

6.37.3.6 update()

```
void udit::Scene::update ()
```

Actualiza la escena.

Actualiza ciertos valores dentro del bucle principal.

Llama a las funciones necesarias para actualizar los objetos en la escena. Este método debe ser llamado cada vez que se desea actualizar el estado de la escena.

The documentation for this class was generated from the following files:

- GL_Scene/[Scene.hpp](#)
- GL_Scene/[Scene.cpp](#)

6.38 Shader Class Reference

Representa un shader program en OpenGL.

```
#include <Shader.hpp>
```

Public Member Functions

- [Shader](#) ()
Constructor por defecto.
- [Shader](#) (ShaderType type, const std::string &vertex_source, const std::string &fragment_source, const std::string &name)
Constructor para crear un shader con tipos y fuentes especificadas.
- [~Shader](#) ()
Destructor.
- GLuint [compile_shaders](#) (const char *vertex_shader_code, const char *fragment_shader_code)
Compila los shaders.
- GLint [get_model_view_matrix_id](#) ()
Obtiene el identificador de la matriz de modelo-vista.
- GLint [get_projection_matrix_id](#) ()
Obtiene el identificador de la matriz de proyección.
- GLint [get_normal_matrix_id](#) ()
Obtiene el identificador de la matriz de normales.
- GLuint [get_program_id](#) () const
Obtiene el identificador del programa de shader.
- void [set_texture](#) (const std::shared_ptr< [Texture](#) > &texture)
Establece una textura para el shader.
- void [use](#) () const
Activa y usa el programa de shader.
- void [set_texture_scale](#) (float scale)
Establece la escala de las texturas asociadas al shader.
- bool [has_textures](#) ()
Verifica si el shader tiene texturas asociadas.
- void [set_name](#) (const std::string &name)
Establece el nombre del shader.
- std::string [get_name](#) ()
Obtiene el nombre del shader.

Static Public Member Functions

- static std::shared_ptr< [Shader](#) > [make_shader](#) ([udit::ShaderType](#) type=[udit::ShaderType::DEFAULT](#), const std::string &vertex_shader="", const std::string &fragment_shader="", const std::vector< std::string > &texture_paths={""}, const std::string &name="")

Crea un shader.

6.38.1 Detailed Description

Representa un shader program en OpenGL.

La clase [Shader](#) gestiona la creación y uso de programas de sombreado en OpenGL. Permite compilar los shaders, vincularlos en un programa y usarlos para renderizar objetos en la escena. También proporciona funciones para gestionar texturas y matrices de transformación, como la matriz de modelo-vista, proyección y normales.

6.38.2 Constructor & Destructor Documentation

6.38.2.1 [Shader\(\)](#) [1/2]

```
udit::Shader::Shader ()
```

Constructor por defecto.

Crea un objeto [Shader](#) sin especificar un tipo o fuentes de shader. Este constructor generalmente se usa para crear shaders más tarde con la función [make_shader](#).

6.38.2.2 [Shader\(\)](#) [2/2]

```
udit::Shader::Shader (
    ShaderType type,
    const std::string & vertex_source,
    const std::string & fragment_source,
    const std::string & name)
```

Constructor para crear un shader con tipos y fuentes especificadas.

Parameters

<i>type</i>	Tipo de shader (e.g., SKYBOX, GEOMETRY).
<i>vertex_source</i>	Código fuente para el vertex shader.
<i>fragment_source</i>	Código fuente para el fragment shader.
<i>name</i>	Nombre del shader.

6.38.2.3 [~Shader\(\)](#)

```
udit::Shader::~Shader ()
```

Destructor.

Libera los recursos asociados al shader.

6.38.3 Member Function Documentation

6.38.3.1 compile_shaders()

```
GLuint udit::Shader::compile_shaders (
    const char * vertex_shader_code,
    const char * fragment_shader_code)
```

Compila los shaders.

Compilador de los shaders construidos.

Compila un vertex shader y un fragment shader usando el código fuente proporcionado.

Parameters

<i>vertex_shader_code</i>	Código fuente del vertex shader.
<i>fragment_shader_code</i>	Código fuente del fragment shader.

Returns

Identificador del programa de shader compilado.

6.38.3.2 get_model_view_matrix_id()

```
GLint udit::Shader::get_model_view_matrix_id () [inline]
```

Obtiene el identificador de la matriz de modelo-vista.

Returns

Identificador de la matriz de modelo-vista.

6.38.3.3 get_name()

```
std::string udit::Shader::get_name () [inline]
```

Obtiene el nombre del shader.

Returns

Nombre del shader.

6.38.3.4 get_normal_matrix_id()

```
GLint udit::Shader::get_normal_matrix_id () [inline]
```

Obtiene el identificador de la matriz de normales.

Returns

Identificador de la matriz de normales.

6.38.3.5 get_program_id()

```
GLuint udit::Shader::get_program_id () const [inline]
```

Obtiene el identificador del programa de shader.

Returns

Identificador del programa de shader.

6.38.3.6 get_projection_matrix_id()

```
GLint udit::Shader::get_projection_matrix_id () [inline]
```

Obtiene el identificador de la matriz de proyección.

Returns

Identificador de la matriz de proyección.

6.38.3.7 has_textures()

```
bool udit::Shader::has_textures () [inline]
```

Verifica si el shader tiene texturas asociadas.

Returns

`true` si el shader tiene texturas asociadas, `false` en caso contrario.

6.38.3.8 make_shader()

```
std::shared_ptr< Shader > udit::Shader::make_shader (
    udit::ShaderType type = udit::ShaderType::DEFAULT,
    const std::string & vertex_shader = "",
    const std::string & fragment_shader = "",
    const std::vector< std::string > & texture_paths = {},
    const std::string & name = "") [static]
```

Crea un shader.

Función estática para crear un shader con un tipo específico y fuentes de shader opcionales.

Parameters

<i>type</i>	Tipo de shader.
<i>vertex_shader</i>	Código fuente del vertex shader.
<i>fragment_shader</i>	Código fuente del fragment shader.
<i>texture_paths</i>	Rutas a las texturas asociadas.
<i>name</i>	Nombre del shader.

Returns

Objeto [Shader](#) creado.

6.38.3.9 set_name()

```
void udit::Shader::set_name (  
    const std::string & name) [inline]
```

Establece el nombre del shader.

Parameters

<i>name</i>	Nombre del shader.
-------------	--------------------

6.38.3.10 set_texture()

```
void udit::Shader::set_texture (
    const std::shared_ptr< Texture > & texture)
```

Establece una textura para el shader.

Parameters

<i>texture</i>	Puntero a la textura que será asignada al shader.
----------------	---

6.38.3.11 set_texture_scale()

```
void udit::Shader::set_texture_scale (
    float scale)
```

Establece la escala de las texturas asociadas al shader.

Parameters

<i>scale</i>	Factor de escala para las texturas.
--------------	-------------------------------------

6.38.3.12 use()

```
void udit::Shader::use () const
```

Activa y usa el programa de shader.

Hace que el programa de shader sea el activo para su uso en la siguiente operación de renderizado.

The documentation for this class was generated from the following files:

- GL_Scene/[Shader.hpp](#)
- GL_Scene/Shader.cpp

6.39 udit::Shader Class Reference

Representa un shader program en OpenGL.

```
#include <Shader.hpp>
```

Public Member Functions

- [Shader](#) ()
Constructor por defecto.
- [Shader](#) ([ShaderType](#) type, const std::string &vertex_source, const std::string &fragment_source, const std::string &name)
Constructor para crear un shader con tipos y fuentes especificadas.
- [~Shader](#) ()
Destructor.
- GLuint [compile_shaders](#) (const char *vertex_shader_code, const char *fragment_shader_code)
Compila los shaders.
- GLint [get_model_view_matrix_id](#) ()
Obtiene el identificador de la matriz de modelo-vista.
- GLint [get_projection_matrix_id](#) ()
Obtiene el identificador de la matriz de proyección.
- GLint [get_normal_matrix_id](#) ()
Obtiene el identificador de la matriz de normales.
- GLuint [get_program_id](#) () const
Obtiene el identificador del programa de shader.
- void [set_texture](#) (const std::shared_ptr< [Texture](#) > &texture)
Establece una textura para el shader.
- void [use](#) () const
Activa y usa el programa de shader.
- void [set_texture_scale](#) (float scale)
Establece la escala de las texturas asociadas al shader.
- bool [has_textures](#) ()
Verifica si el shader tiene texturas asociadas.
- void [set_name](#) (const std::string &name)
Establece el nombre del shader.
- std::string [get_name](#) ()
Obtiene el nombre del shader.

Static Public Member Functions

- static std::shared_ptr< [Shader](#) > [make_shader](#) ([udit::ShaderType](#) type=[udit::ShaderType::DEFAULT](#), const std::string &vertex_shader="", const std::string &fragment_shader="", const std::vector< std::string > &texture_paths={}, const std::string &name="")
Crea un shader.

6.39.1 Detailed Description

Representa un shader program en OpenGL.

La clase [Shader](#) gestiona la creación y uso de programas de sombreado en OpenGL. Permite compilar los shaders, vincularlos en un programa y usarlos para renderizar objetos en la escena. También proporciona funciones para gestionar texturas y matrices de transformación, como la matriz de modelo-vista, proyección y normales.

6.39.2 Constructor & Destructor Documentation

6.39.2.1 Shader() [1/2]

```
udit::Shader::Shader ()
```

Constructor por defecto.

Crea un objeto [Shader](#) sin especificar un tipo o fuentes de shader. Este constructor generalmente se usa para crear shaders más tarde con la función `make_shader`.

6.39.2.2 Shader() [2/2]

```
udit::Shader::Shader (
    ShaderType type,
    const std::string & vertex_source,
    const std::string & fragment_source,
    const std::string & name)
```

Constructor para crear un shader con tipos y fuentes especificadas.

Parameters

<i>type</i>	Tipo de shader (e.g., SKYBOX, GEOMETRY).
<i>vertex_source</i>	Código fuente para el vertex shader.
<i>fragment_source</i>	Código fuente para el fragment shader.
<i>name</i>	Nombre del shader.

6.39.2.3 ~Shader()

```
udit::Shader::~Shader ()
```

Destructor.

Libera los recursos asociados al shader.

6.39.3 Member Function Documentation

6.39.3.1 compile_shaders()

```
GLuint udit::Shader::compile_shaders (
    const char * vertex_shader_code,
    const char * fragment_shader_code)
```

Compila los shaders.

Compilador de los shaders contruidos.

Compila un vertex shader y un fragment shader usando el código fuente proporcionado.

Parameters

<i>vertex_shader_code</i>	Código fuente del vertex shader.
<i>fragment_shader_code</i>	Código fuente del fragment shader.

Returns

Identificador del programa de shader compilado.

6.39.3.2 get_model_view_matrix_id()

```
GLint udit::Shader::get_model_view_matrix_id () [inline]
```

Obtiene el identificador de la matriz de modelo-vista.

Returns

Identificador de la matriz de modelo-vista.

6.39.3.3 get_name()

```
std::string udit::Shader::get_name () [inline]
```

Obtiene el nombre del shader.

Returns

Nombre del shader.

6.39.3.4 get_normal_matrix_id()

```
GLint udit::Shader::get_normal_matrix_id () [inline]
```

Obtiene el identificador de la matriz de normales.

Returns

Identificador de la matriz de normales.

6.39.3.5 get_program_id()

```
GLuint udit::Shader::get_program_id () const [inline]
```

Obtiene el identificador del programa de shader.

Returns

Identificador del programa de shader.

6.39.3.6 get_projection_matrix_id()

```
GLint udit::Shader::get_projection_matrix_id () [inline]
```

Obtiene el identificador de la matriz de proyección.

Returns

Identificador de la matriz de proyección.

6.39.3.7 has_textures()

```
bool udit::Shader::has_textures () [inline]
```

Verifica si el shader tiene texturas asociadas.

Returns

`true` si el shader tiene texturas asociadas, `false` en caso contrario.

6.39.3.8 make_shader()

```
std::shared_ptr< Shader > udit::Shader::make_shader (
    udit::ShaderType type = udit::ShaderType::DEFAULT,
    const std::string & vertex_shader = "",
    const std::string & fragment_shader = "",
    const std::vector< std::string > & texture_paths = {},
    const std::string & name = "") [static]
```

Crea un shader.

Función estática para crear un shader con un tipo específico y fuentes de shader opcionales.

Parameters

<i>type</i>	Tipo de shader.
<i>vertex_shader</i>	Código fuente del vertex shader.
<i>fragment_shader</i>	Código fuente del fragment shader.
<i>texture_paths</i>	Rutas a las texturas asociadas.
<i>name</i>	Nombre del shader.

Returns

Objeto [Shader](#) creado.

6.39.3.9 set_name()

```
void udit::Shader::set_name (
    const std::string & name) [inline]
```

Establece el nombre del shader.

Parameters

<i>name</i>	Nombre del shader.
-------------	--------------------

6.39.3.10 set_texture()

```
void udit::Shader::set_texture (
    const std::shared_ptr< Texture > & texture)
```

Establece una textura para el shader.

Parameters

<i>texture</i>	Puntero a la textura que será asignada al shader.
----------------	---

6.39.3.11 set_texture_scale()

```
void udit::Shader::set_texture_scale (
    float scale)
```

Establece la escala de las texturas asociadas al shader.

Parameters

<i>scale</i>	Factor de escala para las texturas.
--------------	-------------------------------------

6.39.3.12 use()

```
void udit::Shader::use () const
```

Activa y usa el programa de shader.

Hace que el programa de shader sea el activo para su uso en la siguiente operación de renderizado.

The documentation for this class was generated from the following files:

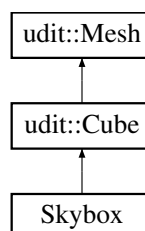
- GL_Scene/[Shader.hpp](#)
- GL_Scene/Shader.cpp

6.40 Skybox Class Reference

Representa un skybox, un cubo con texturas aplicadas en sus seis caras.

```
#include <Skybox.hpp>
```

Inheritance diagram for Skybox:



Public Member Functions

- [Skybox](#) ()
Constructor por defecto.
- [Skybox](#) (float size, const std::vector< std::string > &faces)
Constructor que permite especificar el tamaño y las texturas del skybox.
- unsigned int [getCubemapTexture](#) () const
Obtiene el identificador de la textura cubemap cargada para el skybox.

Public Member Functions inherited from [udit::Cube](#)

- [Cube](#) ()
Constructor por defecto.
- [Cube](#) (bool inverted)
Constructor con opción de invertir las normales.
- [Cube](#) (float size)
Constructor con tamaño especificado.
- [Cube](#) (float size, bool inverted)
Constructor con tamaño y opción de invertir las normales.

Public Member Functions inherited from [udit::Mesh](#)

- [Mesh](#) ()
Constructor por defecto.
- [Mesh](#) (std::string &path)
Constructor que carga una malla desde un archivo.
- virtual [~Mesh](#) ()
Destructor de la clase.
- virtual void [translate](#) (glm::vec3 translation)
Realiza una traslación de la malla.
- virtual void [rotate](#) (glm::vec3 rotation, float angle)
Rota la malla.
- virtual void [scale](#) (glm::vec3 scale)
Escala la malla.
- virtual void [update](#) ()
Actualiza la malla.
- virtual void [render](#) (glm::mat4 view_matrix)
Renderiza la malla.
- virtual void [resize](#) (glm::mat4 projection_matrix)
Ajusta la matriz de proyección.
- virtual void [set_shader](#) (std::shared_ptr< [udit::Shader](#) > shader)
Asocia un shader a la malla.
- GLuint [get_shader_program_id](#) () const
Obtiene el ID del programa del shader asociado.
- std::vector< GLuint > [get_shader_matrix_ids](#) ()
Obtiene los IDs de las matrices del shader asociadas a la malla.
- glm::mat4 [get_model_view_matrix](#) () const
Obtiene la matriz de transformación del modelo.
- void [set_model_view_matrix](#) (glm::mat4 matrix)
Establece la matriz de transformación del modelo.
- void [set_mesh_type](#) (MeshType type)
Establece el tipo de malla.

Additional Inherited Members

Static Public Member Functions inherited from [udit::Mesh](#)

- static `std::shared_ptr< Mesh > make_mesh` (MeshType type, const `std::string` &path="")
Crea una malla de un tipo específico.

Protected Member Functions inherited from [udit::Mesh](#)

- void `create_mesh` (`std::string` mesh_name="")
Crea los VBOs y el VAO necesarios para la malla.

Protected Attributes inherited from [udit::Mesh](#)

- `std::vector< glm::vec3 > coordinates`
Vectores que almacenan las coordenadas de los vértices, colores, normales, índices y coordenadas de textura.
- `std::vector< glm::vec3 > colors`
- `std::vector< glm::vec3 > normals`
- `std::vector< GLuint > indices`
- `std::vector< glm::vec2 > texture_uvs`
- `GLsizei number_of_vertices`
Número total de vértices de la malla.

6.40.1 Detailed Description

Representa un skybox, un cubo con texturas aplicadas en sus seis caras.

Un skybox es un cubo que rodea la escena y sirve como fondo inmersivo en un entorno 3D. La clase [Skybox](#) hereda de la clase [Cube](#), y se encarga de cargar las texturas y mostrar el cielo en una escena utilizando un cubo con caras texturizadas.

6.40.2 Constructor & Destructor Documentation

6.40.2.1 `Skybox()` [1/2]

```
udit::Skybox::Skybox ()
```

Constructor por defecto.

Este constructor crea un skybox con un tamaño por defecto y sin texturas cargadas.

6.40.2.2 `Skybox()` [2/2]

```
udit::Skybox::Skybox (
    float size,
    const std::vector< std::string > & faces)
```

Constructor que permite especificar el tamaño y las texturas del skybox.

Parameters

<i>size</i>	Tamaño del cubo que representará el skybox.
<i>faces</i>	Vector de rutas a las texturas que serán aplicadas a las caras del skybox.

6.40.3 Member Function Documentation

6.40.3.1 getCubemapTexture()

```
unsigned int udit::Skybox::getCubemapTexture () const [inline]
```

Obtiene el identificador de la textura cubemap cargada para el skybox.

Returns

Identificador de la textura cubemap.

The documentation for this class was generated from the following files:

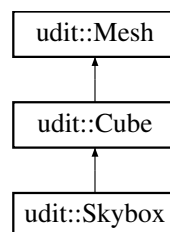
- [GL_Scene/Skybox.hpp](#)
- [GL_Scene/Skybox.cpp](#)

6.41 udit::Skybox Class Reference

Representa un skybox, un cubo con texturas aplicadas en sus seis caras.

```
#include <Skybox.hpp>
```

Inheritance diagram for udit::Skybox:



Public Member Functions

- [Skybox](#) ()
Constructor por defecto.
- [Skybox](#) (float size, const std::vector< std::string > &faces)
Constructor que permite especificar el tamaño y las texturas del skybox.
- unsigned int [getCubemapTexture](#) () const
Obtiene el identificador de la textura cubemap cargada para el skybox.

Public Member Functions inherited from [udit::Cube](#)

- [Cube](#) ()
Constructor por defecto.
- [Cube](#) (bool inverted)
Constructor con opción de invertir las normales.
- [Cube](#) (float size)
Constructor con tamaño especificado.
- [Cube](#) (float size, bool inverted)
Constructor con tamaño y opción de invertir las normales.

Public Member Functions inherited from [udit::Mesh](#)

- [Mesh](#) ()
Constructor por defecto.
- [Mesh](#) (std::string &path)
Constructor que carga una malla desde un archivo.
- virtual [~Mesh](#) ()
Destructor de la clase.
- virtual void [translate](#) (glm::vec3 translation)
Realiza una traslación de la malla.
- virtual void [rotate](#) (glm::vec3 rotation, float angle)
Rota la malla.
- virtual void [scale](#) (glm::vec3 scale)
Escala la malla.
- virtual void [update](#) ()
Actualiza la malla.
- virtual void [render](#) (glm::mat4 view_matrix)
Renderiza la malla.
- virtual void [resize](#) (glm::mat4 projection_matrix)
Ajusta la matriz de proyección.
- virtual void [set_shader](#) (std::shared_ptr< [udit::Shader](#) > shader)
Asocia un shader a la malla.
- GLuint [get_shader_program_id](#) () const
Obtiene el ID del programa del shader asociado.
- std::vector< GLint > [get_shader_matrix_ids](#) ()
Obtiene los IDs de las matrices del shader asociadas a la malla.
- glm::mat4 [get_model_view_matrix](#) () const
Obtiene la matriz de transformación del modelo.
- void [set_model_view_matrix](#) (glm::mat4 matrix)
Establece la matriz de transformación del modelo.
- void [set_mesh_type](#) (MeshType type)
Establece el tipo de malla.

Additional Inherited Members

Static Public Member Functions inherited from [udit::Mesh](#)

- static std::shared_ptr< [Mesh](#) > [make_mesh](#) (MeshType type, const std::string &path="")
Crea una malla de un tipo específico.

Protected Member Functions inherited from [udit::Mesh](#)

- void [create_mesh](#) (std::string mesh_name="")
Crea los VBOs y el VAO necesarios para la malla.

Protected Attributes inherited from [udit::Mesh](#)

- std::vector< glm::vec3 > **coordinates**
Vectores que almacenan las coordenadas de los vértices, colores, normales, índices y coordenadas de textura.
- std::vector< glm::vec3 > **colors**
- std::vector< glm::vec3 > **normals**
- std::vector< GLuint > **indices**
- std::vector< glm::vec2 > **texture_uvs**
- GLsizei **number_of_vertices**
Número total de vértices de la malla.

6.41.1 Detailed Description

Representa un skybox, un cubo con texturas aplicadas en sus seis caras.

Un skybox es un cubo que rodea la escena y sirve como fondo inmersivo en un entorno 3D. La clase [Skybox](#) hereda de la clase [Cube](#), y se encarga de cargar las texturas y mostrar el cielo en una escena utilizando un cubo con caras texturizadas.

6.41.2 Constructor & Destructor Documentation

6.41.2.1 Skybox() [1/2]

```
udit::Skybox::Skybox ()
```

Constructor por defecto.

Este constructor crea un skybox con un tamaño por defecto y sin texturas cargadas.

6.41.2.2 Skybox() [2/2]

```
udit::Skybox::Skybox (
    float size,
    const std::vector< std::string > & faces)
```

Constructor que permite especificar el tamaño y las texturas del skybox.

Parameters

<i>size</i>	Tamaño del cubo que representará el skybox.
<i>faces</i>	Vector de rutas a las texturas que serán aplicadas a las caras del skybox.

6.41.3 Member Function Documentation

6.41.3.1 getCubemapTexture()

```
unsigned int udit::Skybox::getCubemapTexture () const [inline]
```

Obtiene el identificador de la textura cubemap cargada para el skybox.

Returns

Identificador de la textura cubemap.

The documentation for this class was generated from the following files:

- GL_Scene/Skybox.hpp
- GL_Scene/Skybox.cpp

6.42 Texture Class Reference

Representa una textura en OpenGL.

```
#include <Texture.hpp>
```

Public Member Functions

- [Texture](#) (const std::string &path, GLenum [texture_unit](#), Texture_Type type=Texture_Type::COLOR)
Constructor que crea la textura a partir de un archivo.
- [~Texture](#) ()
Destructor que libera la textura cargada.
- void [bind](#) () const
Enlaza la textura a la unidad de textura actual.
- void [unbind](#) () const
Desenlaza la textura de la unidad de textura.
- void [load_texture](#) ()
Carga la textura desde el archivo especificado.
- void [set_type](#) (Texture_Type type)
Establece el tipo de la textura (COLOR o HEIGHT).
- bool [is_loaded](#) ()
Indica si la textura ha sido cargada exitosamente.

Public Attributes

- GLuint [texture_id](#)
Identificador de la textura cargada.
- GLenum [texture_unit](#)
Unidad de textura a la que la textura está asignada.
- std::string [file_path](#)
Ruta del archivo de la textura.

6.42.1 Detailed Description

Representa una textura en OpenGL.

La clase `Texture` permite la carga y manejo de texturas en OpenGL. Estas texturas pueden ser utilizadas en diferentes tipos de materiales y objetos 3D dentro de la escena. La clase gestiona el enlace y des-enlace de texturas, permitiendo su uso en shaders.

6.42.2 Constructor & Destructor Documentation

6.42.2.1 Texture()

```
Texture::Texture (
    const std::string & path,
    GLenum texture_unit,
    Texture_Type type = Texture_Type::COLOR)
```

Constructor que crea la textura a partir de un archivo.

Este constructor carga la textura desde una ruta de archivo específica. Se puede especificar el tipo de textura (por defecto es COLOR).

Parameters

<i>path</i>	Ruta al archivo de la textura (imagen).
<i>texture_unit</i>	Unidad de textura (GL_TEXTURE0, GL_TEXTURE1, etc.).
<i>type</i>	Tipo de la textura (por defecto COLOR).

6.42.3 Member Function Documentation

6.42.3.1 bind()

```
void Texture::bind () const
```

Enlaza la textura a la unidad de textura actual.

Este método enlaza la textura al contexto de OpenGL, permitiendo que sea utilizada por los shaders para renderizar objetos con la textura aplicada.

6.42.3.2 is_loaded()

```
bool udit::Texture::is_loaded () [inline]
```

Indica si la textura ha sido cargada exitosamente.

Returns

true si la textura ha sido cargada, false en caso contrario.

6.42.3.3 load_texture()

```
void Texture::load_texture ()
```

Carga la textura desde el archivo especificado.

Este método lee el archivo de imagen y crea una textura en OpenGL. Se encarga de configurar los parámetros y cargar la imagen a la memoria de GPU.

6.42.3.4 set_type()

```
void udit::Texture::set_type (
    Texture_Type type) [inline]
```

Establece el tipo de la textura (COLOR o HEIGHT).

Parameters

<i>type</i>	Tipo de la textura a establecer.
-------------	----------------------------------

6.42.3.5 unbind()

```
void Texture::unbind () const
```

Desenlaza la textura de la unidad de textura.

Este método desenlaza la textura, liberando la unidad de textura para ser utilizada por otras texturas.

The documentation for this class was generated from the following files:

- [GL_Scene/Texture.hpp](#)
- [GL_Scene/Texture.cpp](#)

6.43 udit::Texture Class Reference

Representa una textura en OpenGL.

```
#include <Texture.hpp>
```

Public Member Functions

- [Texture](#) (const std::string &path, GLenum [texture_unit](#), [Texture_Type](#) type=[Texture_Type::COLOR](#))
Constructor que crea la textura a partir de un archivo.
- [~Texture](#) ()
Destructor que libera la textura cargada.
- void [bind](#) () const
Enlaza la textura a la unidad de textura actual.
- void [unbind](#) () const
Desenlaza la textura de la unidad de textura.
- void [load_texture](#) ()
Carga la textura desde el archivo especificado.
- void [set_type](#) ([Texture_Type](#) type)
Establece el tipo de la textura (COLOR o HEIGHT).
- bool [is_loaded](#) ()
Indica si la textura ha sido cargada exitosamente.

Public Attributes

- GLuint **texture_id**
Identificador de la textura cargada.
- GLenum **texture_unit**
Unidad de textura a la que la textura está asignada.
- std::string **file_path**
Ruta del archivo de la textura.

6.43.1 Detailed Description

Representa una textura en OpenGL.

La clase `Texture` permite la carga y manejo de texturas en OpenGL. Estas texturas pueden ser utilizadas en diferentes tipos de materiales y objetos 3D dentro de la escena. La clase gestiona el enlace y des-enlace de texturas, permitiendo su uso en shaders.

6.43.2 Constructor & Destructor Documentation

6.43.2.1 Texture()

```
Texture::Texture (  
    const std::string & path,  
    GLenum texture_unit,  
    Texture_Type type = Texture_Type::COLOR)
```

Constructor que crea la textura a partir de un archivo.

Este constructor carga la textura desde una ruta de archivo específica. Se puede especificar el tipo de textura (por defecto es COLOR).

Parameters

<i>path</i>	Ruta al archivo de la textura (imagen).
<i>texture_unit</i>	Unidad de textura (GL_TEXTURE0, GL_TEXTURE1, etc.).
<i>type</i>	Tipo de la textura (por defecto COLOR).

6.43.3 Member Function Documentation

6.43.3.1 bind()

```
void Texture::bind () const
```

Enlaza la textura a la unidad de textura actual.

Este método enlaza la textura al contexto de OpenGL, permitiendo que sea utilizada por los shaders para renderizar objetos con la textura aplicada.

6.43.3.2 is_loaded()

```
bool udit::Texture::is_loaded () [inline]
```

Indica si la textura ha sido cargada exitosamente.

Returns

true si la textura ha sido cargada, false en caso contrario.

6.43.3.3 load_texture()

```
void Texture::load_texture ()
```

Carga la textura desde el archivo especificado.

Este método lee el archivo de imagen y crea una textura en OpenGL. Se encarga de configurar los parámetros y cargar la imagen a la memoria de GPU.

6.43.3.4 set_type()

```
void udit::Texture::set_type (  
    Texture_Type type) [inline]
```

Establece el tipo de la textura (COLOR o HEIGHT).

Parameters

<i>type</i>	Tipo de la textura a establecer.
-------------	----------------------------------

6.43.3.5 unbind()

```
void Texture::unbind () const
```

Desenlaza la textura de la unidad de textura.

Este método desenlaza la textura, liberando la unidad de textura para ser utilizada por otras texturas.

The documentation for this class was generated from the following files:

- GL_Scene/[Texture.hpp](#)
- GL_Scene/[Texture.cpp](#)

6.44 udit::Window Class Reference

Classes

- struct [OpenGL_Context_Settings](#)

Public Types

- enum **Position** { **UNDEFINED** = SDL_WINDOWPOS_UNDEFINED , **CENTERED** = SDL_WINDOWPOS_↵
CENTERED }

Public Member Functions

- Window** (const std::string &title, int left_x, int top_y, unsigned width, unsigned height, const [OpenGL_Context_Settings](#) &context_details)
- Window** (const char *title, int left_x, int top_y, unsigned width, unsigned height, const [OpenGL_Context_Settings](#) &context_details)
Constructor de la ventana.
- ~Window** ()
Destructor de la ventana.
- Window** (const [Window](#) &)=delete
- Window & operator=** (const [Window](#) &)=delete
- Window** ([Window](#) &&other) noexcept
- Window & operator=** ([Window](#) &&other) noexcept
- void **swap_buffers** ()
Intercambiar los buffers de OpenGL.

6.44.1 Constructor & Destructor Documentation

6.44.1.1 Window()

```
udit::Window::Window (
    const char * title,
    int left_x,
    int top_y,
    unsigned width,
    unsigned height,
    const OpenGL\_Context\_Settings & context_details)
```

Constructor de la ventana.

Parameters

<i>title</i>	Titulo de la ventana
<i>left_x</i>	Posicion de la ventana en el eje x
<i>top_y</i>	Posicion de la ventana en el eje y
<i>width</i>	Ancho de la ventana
<i>height</i>	Alto de la ventana
<i>context_details</i>	Ajustes el contexto de OpenGL

The documentation for this class was generated from the following files:

- GL_Scene/Window.hpp
- GL_Scene/Window.cpp

6.45 Window Class Reference

Classes

- struct [OpenGL_Context_Settings](#)

Public Types

- enum **Position** { **UNDEFINED** = SDL_WINDOWPOS_UNDEFINED , **CENTERED** = SDL_WINDOWPOS_↵
CENTERED }

Public Member Functions

- **Window** (const std::string &title, int left_x, int top_y, unsigned width, unsigned height, const [OpenGL_Context_Settings](#) &context_details)
- [Window](#) (const char *title, int left_x, int top_y, unsigned width, unsigned height, const [OpenGL_Context_Settings](#) &context_details)
Constructor de la ventana.
- **Window** (const [Window](#) &)=delete
- **Window** ([Window](#) &&other) noexcept
- ~**Window** ()
Destructor de la ventana.
- [Window](#) & **operator=** (const [Window](#) &)=delete
- [Window](#) & **operator=** ([Window](#) &&other) noexcept
- void **swap_buffers** ()
Intercambiar los buffers de OpenGL.

6.45.1 Constructor & Destructor Documentation

6.45.1.1 Window()

```
udit::Window::Window (
    const char * title,
    int left_x,
    int top_y,
    unsigned width,
    unsigned height,
    const OpenGL\_Context\_Settings & context_details)
```

Constructor de la ventana.

Parameters

<i>title</i>	Titulo de la ventana
<i>left_x</i>	Posicion de la ventana en el eje x
<i>top_y</i>	Posicion de la ventana en el eje y
<i>width</i>	Ancho de la ventana
<i>height</i>	Alto de la ventana
<i>context_details</i>	Ajustes el contexto de OpenGL

The documentation for this class was generated from the following files:

- GL_Scene/Window.hpp
- GL_Scene/Window.cpp

Chapter 7

File Documentation

7.1 Camera.hpp

```
00001 //
00002 // Camera.hpp
00003 // GL_Scene
00004 //
00005 // Created by Alonso García on 23/12/24.
00006 //
00007
00008 #pragma once
00009
00010 #include "glm.hpp"
00011 #include <gtc/matrix_transform.hpp>
00012 #include <gtc/constants.hpp>
00013
00014 enum class CameraMovement
00015 {
00016     FORWARD,
00017     BACKWARD,
00018     LEFT,
00019     RIGHT,
00020     UP,
00021     DOWN
00022 };
00023
00024 class Camera
00025 {
00026 public:
00027     glm::vec3 position;
00028
00029     glm::vec3 front;
00030
00031     glm::vec3 up;
00032
00033     glm::vec3 right;
00034
00035     glm::vec3 world_up;
00036
00037     float yaw;
00038
00039     float pitch;
00040
00041     float movement_speed;
00042
00043     float mouse_sensitivity;
00044
00045     float zoom;
00046
00047     Camera(glm::vec3 start_position, glm::vec3 up_direction, float start_yaw, float start_pitch);
00048
00049     glm::mat4 get_view_matrix() const;
00050
00051     void process_keyboard(CameraMovement direction, float delta_time);
00052
00053     void process_mouse_movement(float x_offset, float y_offset, bool constraint_pitch = true);
00054 private:
00055     void update_camera_vectors();
00056 };
```

7.2 Cube.hpp

```

00001 //
00002 //  Cube.hpp
00003 //  GL_Geometry
00004 //
00005 //  Created by Alonso García on 21/12/24.
00006 //
00007
00008 #pragma once
00009
00010 #include "glad.h"
00011
00012 #include "Mesh.hpp"
00013
00022 namespace udit
00023 {
00024     class Cube : public Mesh
00025     {
00026     private:
00032         float size;
00033
00034     public:
00040         Cube();
00041
00050         Cube(bool inverted);
00051
00059         Cube(float size);
00060
00070         Cube(float size, bool inverted);
00071
00072     private:
00081         void create_cube(bool inverted = false);
00082     };
00083 }

```

7.3 EventHandler.hpp

```

00001 //
00002 //  EventHandler.hpp
00003 //  GL_Scene
00004 //
00005 //  Created by Alonso García on 23/12/24.
00006 //
00007
00008 #pragma once
00009
00010 #include "SDL.h"
00011 #include "glm.hpp"
00012 #include "Camera.hpp"
00013
00022 class EventHandler
00023 {
00024 public:
00034     EventHandler(Camera& camera)
00035         : camera(camera), first_mouse(true), last_x(0.0f), last_y(0.0f) {}
00036
00049     void handle_events(bool & running, float delta_time);
00050
00051 private:
00055     Camera & camera;
00056
00063     bool first_mouse;
00064
00068     float last_x;
00069
00073     float last_y;
00074
00084     void process_mouse_motion(const SDL_Event & event);
00085
00097     void process_keyboard(const Uint8 * keystate, float delta_time);
00098 };

```

7.4 GL_Scene/half.hpp File Reference

```

#include <utility>
#include <algorithm>

```

```
#include <istream>
#include <ostream>
#include <limits>
#include <stdexcept>
#include <climits>
#include <cmath>
#include <cstring>
#include <cstdlib>
```

Classes

- struct [half_float::detail::conditional](#)< bool, T, typename >

Conditional type.

- struct [half_float::detail::conditional](#)< false, T, F >
- struct [half_float::detail::bool_type](#)< bool >

Helper for tag dispatching.

- struct [half_float::detail::is_float](#)< typename >

Type traits for floating-point types.

- struct [half_float::detail::is_float](#)< const T >
- struct [half_float::detail::is_float](#)< volatile T >
- struct [half_float::detail::is_float](#)< const volatile T >
- struct [half_float::detail::is_float](#)< float >
- struct [half_float::detail::is_float](#)< double >
- struct [half_float::detail::is_float](#)< long double >
- struct [half_float::detail::bits](#)< T >

Type traits for floating-point bits.

- struct [half_float::detail::bits](#)< const T >
- struct [half_float::detail::bits](#)< volatile T >
- struct [half_float::detail::bits](#)< const volatile T >
- struct [half_float::detail::bits](#)< float >

Unsigned integer of (at least) 32 bits width.

- struct [half_float::detail::bits](#)< double >

Unsigned integer of (at least) 64 bits width.

- struct [half_float::detail::binary_t](#)

Tag type for binary construction.

- struct [half_float::detail::f31](#)

Class for 1.31 unsigned floating-point computation.

- class [half_float::half](#)
- struct [half_float::detail::half_caster](#)< T, U, R >
- struct [half_float::detail::half_caster](#)< half, U, R >
- struct [half_float::detail::half_caster](#)< T, half, R >
- struct [half_float::detail::half_caster](#)< half, half, R >
- class [std::numeric_limits](#)< half_float::half >

Namespaces

- namespace [half_float](#)
- namespace [std](#)

Extensions to the C++ standard library.

Macros

- `#define HALF_GCC_VERSION (__GNUC__*100+__GNUC_MINOR__)`
- `#define HALF_ICC_VERSION 0`
- `#define HALF_ERRHANDLING (HALF_ERRHANDLING_FLAGS||HALF_ERRHANDLING_ERRNO||HALF↵
_ERRHANDLING_FENV||HALF_ERRHANDLING_THROWS)`
- `#define HALF_UNUSED_NOERR(name)`
- `#define HALF_CONSTEXPR`
- `#define HALF_CONSTEXPR_CONST const`
- `#define HALF_CONSTEXPR_NOERR`
- `#define HALF_NOEXCEPT`
- `#define HALF_NOTHROW throw()`
- `#define HALF_THREAD_LOCAL static`
- `#define HALF_ENABLE_F16C_INTRINSICS __F16C__`
- `#define HALF_ERRHANDLING_OVERFLOW_TO_INEXACT 1`
- `#define HALF_ERRHANDLING_UNDERFLOW_TO_INEXACT 1`
- `#define HALF_ROUND_STYLE 1`
- `#define HUGE_VALH std::numeric_limits<half_float::half>::infinity()`
- `#define FP_FAST_FMAH 1`
- `#define HLF_ROUNDS HALF_ROUND_STYLE`
- `#define FP_ILOGB0 INT_MIN`
- `#define FP_ILOGBNAN INT_MAX`
- `#define FP_SUBNORMAL 0`
- `#define FP_ZERO 1`
- `#define FP_NAN 2`
- `#define FP_INFINITE 3`
- `#define FP_NORMAL 4`
- `#define FE_INVALID 0x10`
- `#define FE_DIVBYZERO 0x08`
- `#define FE_OVERFLOW 0x04`
- `#define FE_UNDERFLOW 0x02`
- `#define FE_INEXACT 0x01`
- `#define FE_ALL_EXCEPT (FE_INVALID|FE_DIVBYZERO|FE_OVERFLOW|FE_UNDERFLOW|FE_↵
INEXACT)`

Typedefs

- `typedef bool_type< true > half_float::detail::true_type`
- `typedef bool_type< false > half_float::detail::false_type`
- `typedef unsigned short half_float::detail::uint16`
Unsigned integer of (at least) 16 bits width.
- `typedef unsigned long half_float::detail::uint32`
Fastest unsigned integer of (at least) 32 bits width.
- `typedef long half_float::detail::int32`
Fastest unsigned integer of (at least) 32 bits width.

Functions

Implementation defined classification and arithmetic

- `template<typename T>`
`bool half_float::detail::builtin_isinf (T arg)`
- `template<typename T>`
`bool half_float::detail::builtin_isnan (T arg)`
- `template<typename T>`
`bool half_float::detail::builtin_signbit (T arg)`
- `uint32 half_float::detail::sign_mask (uint32 arg)`
- `uint32 half_float::detail::arithmetic_shift (uint32 arg, int i)`

Error handling

- `int & half_float::detail::errflags ()`
- `void half_float::detail::raise (int HALF_UNUSED_NOERR(flags), bool HALF_UNUSED_NOERR(cond)=true)`
- `HALF_CONSTEXPR_NOERR bool half_float::detail::compsignal (unsigned int x, unsigned int y)`
- `HALF_CONSTEXPR_NOERR unsigned int half_float::detail::signal (unsigned int nan)`
- `HALF_CONSTEXPR_NOERR unsigned int half_float::detail::signal (unsigned int x, unsigned int y)`
- `HALF_CONSTEXPR_NOERR unsigned int half_float::detail::signal (unsigned int x, unsigned int y, unsigned int z)`
- `HALF_CONSTEXPR_NOERR unsigned int half_float::detail::select (unsigned int x, unsigned int HALF_UNUSED_NOERR(y))`
- `HALF_CONSTEXPR_NOERR unsigned int half_float::detail::invalid ()`
- `HALF_CONSTEXPR_NOERR unsigned int half_float::detail::pole (unsigned int sign=0)`
- `HALF_CONSTEXPR_NOERR unsigned int half_float::detail::check_underflow (unsigned int arg)`

Conversion and rounding

- `template<std::float_round_style R>`
`HALF_CONSTEXPR_NOERR unsigned int half_float::detail::overflow (unsigned int sign=0)`
- `template<std::float_round_style R>`
`HALF_CONSTEXPR_NOERR unsigned int half_float::detail::underflow (unsigned int sign=0)`
- `template<std::float_round_style R, bool I>`
`HALF_CONSTEXPR_NOERR unsigned int half_float::detail::rounded (unsigned int value, int g, int s)`
- `template<std::float_round_style R, bool E, bool I>`
`unsigned int half_float::detail::integral (unsigned int value)`
- `template<std::float_round_style R, unsigned int F, bool S, bool N, bool I>`
`unsigned int half_float::detail::fixed2half (uint32 m, int exp=14, unsigned int sign=0, int s=0)`
- `template<std::float_round_style R>`
`unsigned int half_float::detail::float2half_impl (float value, true_type)`
- `template<std::float_round_style R>`
`unsigned int half_float::detail::float2half_impl (double value, true_type)`
- `template<std::float_round_style R, typename T>`
`unsigned int half_float::detail::float2half_impl (T value,...)`
- `template<std::float_round_style R, typename T>`
`unsigned int half_float::detail::float2half (T value)`
- `template<std::float_round_style R, typename T>`
`unsigned int half_float::detail::int2half (T value)`
- `float half_float::detail::half2float_impl (unsigned int value, float, true_type)`
- `double half_float::detail::half2float_impl (unsigned int value, double, true_type)`
- `template<typename T>`
`T half_float::detail::half2float_impl (unsigned int value, T,...)`
- `template<typename T>`
`T half_float::detail::half2float (unsigned int value)`
- `template<std::float_round_style R, bool E, bool I, typename T>`
`T half_float::detail::half2int (unsigned int value)`

Mathematics

- `template<std::float_round_style R>`
`uint32 half_float::detail::mulhi (uint32 x, uint32 y)`
- `uint32 half_float::detail::multiply64 (uint32 x, uint32 y)`
- `uint32 half_float::detail::divide64 (uint32 x, uint32 y, int &s)`
- `template<bool Q, bool R>`
`unsigned int half_float::detail::mod (unsigned int x, unsigned int y, int *quo=NULL)`
- `template<unsigned int F>`
`uint32 half_float::detail::sqrt (uint32 &r, int &exp)`
- `uint32 half_float::detail::exp2 (uint32 m, unsigned int n=32)`
- `uint32 half_float::detail::log2 (uint32 m, unsigned int n=32)`
- `std::pair< uint32, uint32 > half_float::detail::sincos (uint32 mz, unsigned int n=31)`
- `uint32 half_float::detail::atan2 (uint32 my, uint32 mx, unsigned int n=31)`
- `uint32 half_float::detail::angle_arg (unsigned int abs, int &k)`
- `std::pair< uint32, uint32 > half_float::detail::atan2_args (unsigned int abs)`
- `std::pair< uint32, uint32 > half_float::detail::hyperbolic_args (unsigned int abs, int &exp, unsigned int n=32)`
- `template<std::float_round_style R>`
`unsigned int half_float::detail::exp2_post (uint32 m, int exp, bool esign, unsigned int sign=0, unsigned int n=32)`
- `template<std::float_round_style R, uint32 L>`
`unsigned int half_float::detail::log2_post (uint32 m, int ilog, int exp, unsigned int sign=0)`
- `template<std::float_round_style R>`
`unsigned int half_float::detail::hypot_post (uint32 r, int exp)`
- `template<std::float_round_style R>`
`unsigned int half_float::detail::tangent_post (uint32 my, uint32 mx, int exp, unsigned int sign=0)`
- `template<std::float_round_style R, bool S>`
`unsigned int half_float::detail::area (unsigned int arg)`
- `template<std::float_round_style R, bool C>`
`unsigned int half_float::detail::erf (unsigned int arg)`
- `template<std::float_round_style R, bool L>`
`unsigned int half_float::detail::gamma (unsigned int arg)`

Comparison operators

- `HALF_CONSTEXPR_NOERR bool half_float::operator== (half x, half y)`
- `HALF_CONSTEXPR_NOERR bool half_float::operator!= (half x, half y)`
- `HALF_CONSTEXPR_NOERR bool half_float::operator< (half x, half y)`
- `HALF_CONSTEXPR_NOERR bool half_float::operator> (half x, half y)`
- `HALF_CONSTEXPR_NOERR bool half_float::operator<= (half x, half y)`
- `HALF_CONSTEXPR_NOERR bool half_float::operator>= (half x, half y)`

Arithmetic operators

- `HALF_CONSTEXPR half half_float::operator+ (half arg)`
- `HALF_CONSTEXPR half half_float::operator- (half arg)`
- `half half_float::operator+ (half x, half y)`
- `half half_float::operator- (half x, half y)`
- `half half_float::operator* (half x, half y)`
- `half half_float::operator/ (half x, half y)`

Input and output

- `template<typename charT, typename traits>`
`std::basic_ostream< charT, traits > & half_float::operator<< (std::basic_ostream< charT, traits > &out, half arg)`
- `template<typename charT, typename traits>`
`std::basic_istream< charT, traits > & half_float::operator>> (std::basic_istream< charT, traits > &in, half &arg)`

Basic mathematical operations

- `half CONSTEXPR half half_float::fabs (half arg)`
- `half CONSTEXPR half half_float::abs (half arg)`
- `half half_float::fmod (half x, half y)`
- `half half_float::remainder (half x, half y)`
- `half half_float::remquo (half x, half y, int *quo)`
- `half half_float::fma (half x, half y, half z)`
- `half CONSTEXPR_NOERR half half_float::fmax (half x, half y)`
- `half CONSTEXPR_NOERR half half_float::fmin (half x, half y)`
- `half half_float::fdim (half x, half y)`
- `half half_float::nanh (const char *arg)`

Exponential functions

- `half half_float::exp (half arg)`
- `half half_float::exp2 (half arg)`
- `half half_float::expm1 (half arg)`
- `half half_float::log (half arg)`
- `half half_float::log10 (half arg)`
- `half half_float::log2 (half arg)`
- `half half_float::log1p (half arg)`

Power functions

- `half half_float::sqrt (half arg)`
- `half half_float::rsqrt (half arg)`
- `half half_float::cbrt (half arg)`
- `half half_float::hypot (half x, half y)`
- `half half_float::hypot (half x, half y, half z)`
- `half half_float::pow (half x, half y)`

Trigonometric functions

- `void half_float::sincos (half arg, half *sin, half *cos)`
- `half half_float::sin (half arg)`
- `half half_float::cos (half arg)`
- `half half_float::tan (half arg)`
- `half half_float::asin (half arg)`
- `half half_float::acos (half arg)`
- `half half_float::atan (half arg)`
- `half half_float::atan2 (half y, half x)`

Hyperbolic functions

- `half half_float::sinh (half arg)`
- `half half_float::cosh (half arg)`
- `half half_float::tanh (half arg)`
- `half half_float::asinh (half arg)`
- `half half_float::acosh (half arg)`
- `half half_float::atanh (half arg)`

Error and gamma functions

- `half half_float::erf (half arg)`
- `half half_float::erfc (half arg)`
- `half half_float::lgamma (half arg)`
- `half half_float::tgamma (half arg)`

Rounding

- `half half_float::ceil (half arg)`
- `half half_float::floor (half arg)`
- `half half_float::trunc (half arg)`
- `half half_float::round (half arg)`
- `long half_float::lround (half arg)`
- `half half_float::rint (half arg)`
- `long half_float::lrint (half arg)`
- `half half_float::nearbyint (half arg)`

Floating point manipulation

- `half half_float::frexp (half arg, int *exp)`
- `half half_float::scalbln (half arg, long exp)`
- `half half_float::scalbn (half arg, int exp)`
- `half half_float::ldexp (half arg, int exp)`
- `half half_float::modf (half arg, half *iptr)`
- `int half_float::ilogb (half arg)`
- `half half_float::logb (half arg)`
- `half half_float::nextafter (half from, half to)`
- `half half_float::nexttoward (half from, long double to)`
- `HALF_CONSTEXPR half half_float::copysign (half x, half y)`

Floating point classification

- `HALF_CONSTEXPR int half_float::fpclassify (half arg)`
- `HALF_CONSTEXPR bool half_float::isfinite (half arg)`
- `HALF_CONSTEXPR bool half_float::isinf (half arg)`
- `HALF_CONSTEXPR bool half_float::isnan (half arg)`
- `HALF_CONSTEXPR bool half_float::isnormal (half arg)`
- `HALF_CONSTEXPR bool half_float::signbit (half arg)`

Comparison

- `HALF_CONSTEXPR bool half_float::isgreater (half x, half y)`
- `HALF_CONSTEXPR bool half_float::isgreaterequal (half x, half y)`
- `HALF_CONSTEXPR bool half_float::isless (half x, half y)`
- `HALF_CONSTEXPR bool half_float::islessequal (half x, half y)`
- `HALF_CONSTEXPR bool half_float::islessgreater (half x, half y)`
- `HALF_CONSTEXPR bool half_float::isunordered (half x, half y)`

Casting

- `template<typename T, typename U>
T half_float::half_cast (U arg)`
- `template<typename T, std::float_round_style R, typename U>
T half_float::half_cast (U arg)`

Error handling

- `int half_float::feclearexcept (int excepts)`
- `int half_float::fetestexcept (int excepts)`
- `int half_float::feraiseexcept (int excepts)`
- `int half_float::fegetexceptflag (int *flagp, int excepts)`
- `int half_float::fesetexceptflag (const int *flagp, int excepts)`
- `void half_float::fethrowexcept (int excepts, const char *msg="")`

Variables

- HALF_CONSTEXPR_CONST [binary_t](#) `half_float::detail::binary = binary_t()`
Tag for binary construction.

7.4.1 Detailed Description

Main header file for half-precision functionality.

7.4.2 Macro Definition Documentation

7.4.2.1 FP_FAST_FMAH

```
#define FP_FAST_FMAH 1
```

Fast half-precision fma function. This symbol is defined if the fma() function generally executes as fast as, or faster than, a separate half-precision multiplication followed by an addition, which is always the case.

See also: Documentation for [FP_FAST_FMA](#)

7.4.2.2 HALF_ENABLE_F16C_INTRINSICS

```
#define HALF_ENABLE_F16C_INTRINSICS __F16C__
```

Enable F16C instruction set intrinsics. Defining this to 1 enables the use of [F16C compiler intrinsics](#) for converting between half-precision and single-precision values which may result in improved performance. This will not perform additional checks for support of the F16C instruction set, so an appropriate target platform is required when enabling this feature.

Unless predefined it will be enabled automatically when the `__F16C__` symbol is defined, which some compilers do on supporting platforms.

7.4.2.3 HALF_ERRHANDLING_OVERFLOW_TO_INEXACT

```
#define HALF_ERRHANDLING_OVERFLOW_TO_INEXACT 1
```

Raise INEXACT exception on overflow. Defining this to 1 (default) causes overflow errors to automatically raise inexact exceptions in addition. These will be raised after any possible handling of the underflow exception.

7.4.2.4 HALF_ERRHANDLING_UNDERFLOW_TO_INEXACT

```
#define HALF_ERRHANDLING_UNDERFLOW_TO_INEXACT 1
```

Raise INEXACT exception on underflow. Defining this to 1 (default) causes underflow errors to automatically raise inexact exceptions in addition. These will be raised after any possible handling of the underflow exception.

Note: This will actually cause underflow (and the accompanying inexact) exceptions to be raised *only* when the result is inexact, while if disabled bare underflow errors will be raised for *any* (possibly exact) subnormal result.

7.4.2.5 HALF_ROUND_STYLE

```
#define HALF_ROUND_STYLE 1
```

Default rounding mode. This specifies the rounding mode used for all conversions between [halves](#) and more precise types (unless using `half_cast()` and specifying the rounding mode directly) as well as in arithmetic operations and mathematical functions. It can be redefined (before including [half.hpp](#)) to one of the standard rounding modes using their respective constants or the equivalent values of [std::float_round_style](#):

<code>std::float_round_style</code>	value	rounding
<code>std::round_indeterminate</code>	-1	fastest
<code>std::round_toward_zero</code>	0	toward zero
<code>std::round_to_nearest</code>	1	to nearest (default)
<code>std::round_toward_infinity</code>	2	toward positive infinity
<code>std::round_toward_neg_infinity</code>	3	toward negative infinity

By default this is set to 1 (`std::round_to_nearest`), which rounds results to the nearest representable value. It can even be set to `std::numeric_limits<float>::round_style` to synchronize the rounding mode with that of the built-in single-precision implementation (which is likely `std::round_to_nearest`, though).

7.4.2.6 HLF_ROUNDS

```
#define HLF_ROUNDS HALF_ROUND_STYLE
```

Half rounding mode. In correspondence with `FLT_ROUNDS` from `<float>` this symbol expands to the rounding mode used for half-precision operations. It is an alias for `HALF_ROUND_STYLE`.

See also: Documentation for `FLT_ROUNDS`

7.4.2.7 HUGE_VALH

```
#define HUGE_VALH std::numeric_limits<half_float::half>::infinity()
```

Value signaling overflow. In correspondence with `HUGE_VAL[F|L]` from `<cmath>` this symbol expands to a positive value signaling the overflow of an operation, in particular it just evaluates to positive infinity.

See also: Documentation for `HUGE_VAL`

7.4.3 Function Documentation

7.4.3.1 angle_arg()

```
uint32 half_float::detail::angle_arg (
    unsigned int abs,
    int & k) [inline]
```

Reduce argument for trigonometric functions.

Parameters

<i>abs</i>	half-precision floating-point value
<i>k</i>	value to take quarter period

Returns

abs reduced to $[-\pi/4, \pi/4]$ as Q0.30

7.4.3.2 area()

```
template<std::float_round_style R, bool S>
unsigned int half_float::detail::area (
    unsigned int arg)
```

Area function and postprocessing. This computes the value directly in Q2.30 using the representation $\operatorname{asinh}|\operatorname{acosh}(x)| = \log(x + \sqrt{x^2 + 1})$.

Template Parameters

<i>R</i>	rounding mode to use
<i>S</i>	true for asinh, false for acosh

Parameters

<i>arg</i>	half-precision argument
------------	-------------------------

Returns

asinh|acosh(*arg*) converted to half-precision

Exceptions

<i>FE_OVERFLOW</i>	on overflows
<i>FE_UNDERFLOW</i>	on underflows
<i>FE_INEXACT</i>	if no other exception occurred

7.4.3.3 arithmetic_shift()

```
uint32 half_float::detail::arithmetic_shift (
    uint32 arg,
    int i) [inline]
```

Platform-independent arithmetic right shift.

Parameters

<i>arg</i>	integer value in two's complement
<i>i</i>	shift amount (at most 31)

Returns

arg right shifted for *i* bits with possible sign extension

7.4.3.4 atan2()

```
uint32 half_float::detail::atan2 (
    uint32 my,
    uint32 mx,
    unsigned int n = 31) [inline]
```

Fixed point arc tangent. This uses the CORDIC algorithm in vectoring mode.

Parameters

<i>my</i>	y coordinate as Q0.30
<i>mx</i>	x coordinate as Q0.30
<i>n</i>	number of iterations (at most 31)

Returns

arc tangent of *my* / *mx* as Q1.30

7.4.3.5 atan2_args()

```
std::pair< uint32, uint32 > half_float::detail::atan2_args (
    unsigned int abs) [inline]
```

Get arguments for atan2 function.

Parameters

<i>abs</i>	half-precision floating-point value
------------	-------------------------------------

Returns

abs and $\sqrt{1 - abs^2}$ as Q0.30

7.4.3.6 builtin_isinf()

```
template<typename T>
bool half_float::detail::builtin_isinf (
    T arg)
```

Check for infinity.

Template Parameters

<i>T</i>	argument type (builtin floating-point type)
----------	---

Parameters

<i>arg</i>	value to query
------------	----------------

Return values

<i>true</i>	if infinity
<i>false</i>	else

7.4.3.7 builtin_isnan()

```
template<typename T>
bool half_float::detail::builtin_isnan (
    T arg)
```

Check for NaN.

Template Parameters

<i>T</i>	argument type (builtin floating-point type)
----------	---

Parameters

<i>arg</i>	value to query
------------	----------------

Return values

<i>true</i>	if not a number
<i>false</i>	else

7.4.3.8 builtin_signbit()

```
template<typename T>
bool half_float::detail::builtin_signbit (
    T arg)
```

Check sign.

Template Parameters

<i>T</i>	argument type (builtin floating-point type)
----------	---

Parameters

<i>arg</i>	value to query
------------	----------------

Return values

<i>true</i>	if signbit set
<i>false</i>	else

7.4.3.9 check_underflow()

```
HALF_CONSTEXPR_NOERR unsigned int half_float::detail::check_underflow (
    unsigned int arg) [inline]
```

Check value for underflow.

Parameters

<i>arg</i>	non-zero half-precision value to check
------------	--

Returns

arg

Exceptions

<i>FE_UNDERFLOW</i>	if arg is subnormal
---------------------	---------------------

7.4.3.10 compsignal()

```
HALF_CONSTEXPR_NOERR bool half_float::detail::compsignal (
    unsigned int x,
    unsigned int y) [inline]
```

Check and signal for any NaN.

Parameters

<i>x</i>	first half-precision value to check
<i>y</i>	second half-precision value to check

Return values

<i>true</i>	if either <i>x</i> or <i>y</i> is NaN
<i>false</i>	else

Exceptions

<i>FE_INVALID</i>	if <i>x</i> or <i>y</i> is NaN
-------------------	--------------------------------

7.4.3.11 divide64()

```
uint32 half_float::detail::divide64 (
    uint32 x,
    uint32 y,
    int & s) [inline]
```

64-bit division.

Parameters

<i>x</i>	upper 32 bit of dividend
<i>y</i>	divisor
<i>s</i>	variable to store sticky bit for rounding

Returns

$(x \ll 32) / y$

7.4.3.12 erf()

```
template<std::float_round_style R, bool C>
unsigned int half_float::detail::erf (
    unsigned int arg)
```

Error function and postprocessing. This computes the value directly in Q1.31 using the approximations given [here](#).

Template Parameters

<i>R</i>	rounding mode to use
<i>C</i>	<code>true</code> for complementary error function, <code>false</code> else

Parameters

<i>arg</i>	half-precision function argument
------------	----------------------------------

Returns

approximated value of error function in half-precision

Exceptions

<i>FE_OVERFLOW</i>	on overflows
<i>FE_UNDERFLOW</i>	on underflows
<i>FE_INEXACT</i>	if no other exception occurred

7.4.3.13 `errflags()`

```
int & half_float::detail::errflags () [inline]
```

Internal exception flags.

Returns

reference to global exception flags

7.4.3.14 `exp2()`

```
uint32 half_float::detail::exp2 (
    uint32 m,
    unsigned int n = 32) [inline]
```

Fixed point binary exponential. This uses the BKM algorithm in E-mode.

Parameters

<i>m</i>	exponent in [0,1) as Q0.31
<i>n</i>	number of iterations (at most 32)

Returns

2^m as Q1.31

7.4.3.15 exp2_post()

```
template<std::float_round_style R>
unsigned int half_float::detail::exp2_post (
    uint32 m,
    int exp,
    bool esign,
    unsigned int sign = 0,
    unsigned int n = 32)
```

Postprocessing for binary exponential.

Template Parameters

<i>R</i>	rounding mode to use
----------	----------------------

Parameters

<i>m</i>	fractional part of as Q0.31
<i>exp</i>	absolute value of unbiased exponent
<i>esign</i>	sign of actual exponent
<i>sign</i>	sign bit of result
<i>n</i>	number of BKM iterations (at most 32)

Returns

value converted to half-precision

Exceptions

<i>FE_OVERFLOW</i>	on overflows
<i>FE_UNDERFLOW</i>	on underflows
<i>FE_INEXACT</i>	if value had to be rounded or <i>I</i> is <code>true`</code>

7.4.3.16 fixed2half()

```
template<std::float_round_style R, unsigned int F, bool S, bool N, bool I>
unsigned int half_float::detail::fixed2half (
    uint32 m,
    int exp = 14,
    unsigned int sign = 0,
    int s = 0)
```

Convert fixed point to half-precision floating-point.

Template Parameters

<i>R</i>	rounding mode to use
<i>F</i>	number of fractional bits in [11,31]
<i>S</i>	<code>true</code> for signed, <code>false</code> for unsigned
<i>N</i>	<code>true</code> for additional normalization step, <code>false</code> if already normalized to 1.F
<i>I</i>	<code>true</code> to always raise INEXACT exception, <code>false</code> to raise only for rounded results

Parameters

<i>m</i>	mantissa in Q1.F fixed point format
<i>exp</i>	biased exponent - 1
<i>sign</i>	half-precision value with sign bit only
<i>s</i>	sticky bit (or of all but the most significant already discarded bits)

Returns

value converted to half-precision

Exceptions

<i>FE_OVERFLOW</i>	on overflows
<i>FE_UNDERFLOW</i>	on underflows
<i>FE_INEXACT</i>	if value had to be rounded or <i>l</i> is <code>`true`</code>

7.4.3.17 float2half()

```
template<std::float_round_style R, typename T>
unsigned int half_float::detail::float2half (
    T value)
```

Convert floating-point to half-precision.

Template Parameters

<i>R</i>	rounding mode to use
<i>T</i>	source type (builtin floating-point type)

Parameters

<i>value</i>	floating-point value to convert
--------------	---------------------------------

Returns

rounded half-precision value

Exceptions

<i>FE_OVERFLOW</i>	on overflows
<i>FE_UNDERFLOW</i>	on underflows
<i>FE_INEXACT</i>	if value had to be rounded

7.4.3.18 float2half_impl() [1/3]

```
template<std::float_round_style R>
unsigned int half_float::detail::float2half_impl (
    double value,
    true_type )
```

Convert IEEE double-precision to half-precision.

Template Parameters

<i>R</i>	rounding mode to use
----------	----------------------

Parameters

<i>value</i>	double-precision value to convert
--------------	-----------------------------------

Returns

rounded half-precision value

Exceptions

<i>FE_OVERFLOW</i>	on overflows
<i>FE_UNDERFLOW</i>	on underflows
<i>FE_INEXACT</i>	if value had to be rounded

7.4.3.19 float2half_impl() [2/3]

```
template<std::float_round_style R>
unsigned int half_float::detail::float2half_impl (
    float value,
    true_type )
```

Convert IEEE single-precision to half-precision. Credit for this goes to [Jeroen van der Zijp](#).

Template Parameters

<i>R</i>	rounding mode to use
----------	----------------------

Parameters

<i>value</i>	single-precision value to convert
--------------	-----------------------------------

Returns

rounded half-precision value

Exceptions

<i>FE_OVERFLOW</i>	on overflows
<i>FE_UNDERFLOW</i>	on underflows
<i>FE_INEXACT</i>	if value had to be rounded

7.4.3.20 float2half_impl() [3/3]

```
template<std::float_round_style R, typename T>
unsigned int half_float::detail::float2half_impl (
    T value,
    ...)
```

Convert non-IEEE floating-point to half-precision.

Template Parameters

<i>R</i>	rounding mode to use
<i>T</i>	source type (builtin floating-point type)

Parameters

<i>value</i>	floating-point value to convert
--------------	---------------------------------

Returns

rounded half-precision value

Exceptions

<i>FE_OVERFLOW</i>	on overflows
<i>FE_UNDERFLOW</i>	on underflows
<i>FE_INEXACT</i>	if value had to be rounded

7.4.3.21 **gamma()**

```
template<std::float_round_style R, bool L>
unsigned int half_float::detail::gamma (
    unsigned int arg)
```

Gamma function and postprocessing. This approximates the value of either the gamma function or its logarithm directly in Q1.31.

Template Parameters

<i>R</i>	rounding mode to use
<i>L</i>	<code>true</code> for lograithm of gamma function, <code>false</code> for gamma function

Parameters

<i>arg</i>	half-precision floating-point value
------------	-------------------------------------

Returns

lgamma/tgamma(*arg*) in half-precision

Exceptions

<i>FE_OVERFLOW</i>	on overflows
<i>FE_UNDERFLOW</i>	on underflows
<i>FE_INEXACT</i>	if <i>arg</i> is not a positive integer

7.4.3.22 half2float()

```
template<typename T>
T half_float::detail::half2float (
    unsigned int value)
```

Convert half-precision to floating-point.

Template Parameters

<i>T</i>	type to convert to (builtin integer type)
----------	---

Parameters

<i>value</i>	half-precision value to convert
--------------	---------------------------------

Returns

floating-point value

7.4.3.23 half2float_impl() [1/3]

```
double half_float::detail::half2float_impl (
    unsigned int value,
    double ,
    true_type ) [inline]
```

Convert half-precision to IEEE double-precision.

Parameters

<i>value</i>	half-precision value to convert
--------------	---------------------------------

Returns

double-precision value

7.4.3.24 half2float_impl() [2/3]

```
float half_float::detail::half2float_impl (
    unsigned int value,
    float ,
    true_type ) [inline]
```

Convert half-precision to IEEE single-precision. Credit for this goes to [Jeroen van der Zijp](#).

Parameters

<i>value</i>	half-precision value to convert
--------------	---------------------------------

Returns

single-precision value

7.4.3.25 half2float_impl() [3/3]

```
template<typename T>
T half_float::detail::half2float_impl (
    unsigned int value,
    T ,
    ...)

```

Convert half-precision to non-IEEE floating-point.

Template Parameters

<i>T</i>	type to convert to (builtin integer type)
----------	---

Parameters

<i>value</i>	half-precision value to convert
--------------	---------------------------------

Returns

floating-point value

7.4.3.26 half2int()

```
template<std::float_round_style R, bool E, bool I, typename T>
T half_float::detail::half2int (
    unsigned int value)
```

Convert half-precision floating-point to integer.

Template Parameters

<i>R</i>	rounding mode to use
<i>E</i>	<code>true</code> for round to even, <code>false</code> for round away from zero
<i>I</i>	<code>true</code> to raise INEXACT exception (if inexact), <code>false</code> to never raise it
<i>T</i>	type to convert to (builtin integer type with at least 16 bits precision, excluding any implicit sign bits)

Parameters

<i>value</i>	half-precision value to convert
--------------	---------------------------------

Returns

rounded integer value

Exceptions

<i>FE_INVALID</i>	if value is not representable in type <i>T</i>
<i>FE_INEXACT</i>	if value had to be rounded and <i>I</i> is <code>true</code>

7.4.3.27 hyperbolic_args()

```
std::pair< uint32, uint32 > half_float::detail::hyperbolic_args (
    unsigned int abs,
    int & exp,
    unsigned int n = 32) [inline]
```

Get exponentials for hyperbolic computation

Parameters

<i>abs</i>	half-precision floating-point value
<i>exp</i>	variable to take unbiased exponent of larger result
<i>n</i>	number of BKM iterations (at most 32)

Returns

$\exp(abs)$ and $\exp(-abs)$ as Q1.31 with same exponent

7.4.3.28 hypot_post()

```
template<std::float_round_style R>
unsigned int half_float::detail::hypot_post (
    uint32 r,
    int exp)
```

Hypotenuse square root and postprocessing.

Template Parameters

<i>R</i>	rounding mode to use
----------	----------------------

Parameters

<i>r</i>	mantissa as Q2.30
<i>exp</i>	biased exponent

Returns

square root converted to half-precision

Exceptions

<i>FE_OVERFLOW</i>	on overflows
<i>FE_UNDERFLOW</i>	on underflows
<i>FE_INEXACT</i>	if value had to be rounded

7.4.3.29 int2half()

```
template<std::float_round_style R, typename T>
unsigned int half_float::detail::int2half (
    T value)
```

Convert integer to half-precision floating-point.

Template Parameters

<i>R</i>	rounding mode to use
<i>T</i>	type to convert (builtin integer type)

Parameters

<i>value</i>	integral value to convert
--------------	---------------------------

Returns

rounded half-precision value

Exceptions

<i>FE_OVERFLOW</i>	on overflows
<i>FE_INEXACT</i>	if value had to be rounded

7.4.3.30 `integral()`

```
template<std::float_round_style R, bool E, bool I>
unsigned int half_float::detail::integral (
    unsigned int value)
```

Round half-precision number to nearest integer value.

Template Parameters

<i>R</i>	rounding mode to use
<i>E</i>	<code>true</code> for round to even, <code>false</code> for round away from zero
<i>I</i>	<code>true</code> to raise INEXACT exception (if inexact), <code>false</code> to never raise it

Parameters

<i>value</i>	half-precision value to round
--------------	-------------------------------

Returns

half-precision bits for nearest integral value

Exceptions

<i>FE_INVALID</i>	for signaling NaN
<i>FE_INEXACT</i>	if value had to be rounded and <i>I</i> is <code>true</code>

7.4.3.31 `invalid()`

```
HALF_CONSTEXPR_NOERR unsigned int half_float::detail::invalid () [inline]
```

Raise domain error and return NaN. return quiet NaN

Exceptions

<i>FE_INVALID</i>	
-------------------	--

7.4.3.32 `log2()`

```
uint32 half_float::detail::log2 (
    uint32 m,
    unsigned int n = 32) [inline]
```

Fixed point binary logarithm. This uses the BKM algorithm in L-mode.

Parameters

<i>m</i>	mantissa in [1,2) as Q1.30
<i>n</i>	number of iterations (at most 32)

Returns

$\log_2(m)$ as Q0.31

7.4.3.33 `log2_post()`

```
template<std::float_round_style R, uint32 L>
unsigned int half_float::detail::log2_post (
    uint32 m,
    int ilog,
    int exp,
    unsigned int sign = 0)
```

Postprocessing for binary logarithm.

Template Parameters

<i>R</i>	rounding mode to use
<i>L</i>	logarithm for base transformation as Q1.31

Parameters

<i>m</i>	fractional part of logarithm as Q0.31
<i>ilog</i>	signed integer part of logarithm
<i>exp</i>	biased exponent of result
<i>sign</i>	sign bit of result

Returns

value base-transformed and converted to half-precision

Exceptions

<i>FE_OVERFLOW</i>	on overflows
<i>FE_UNDERFLOW</i>	on underflows
<i>FE_INEXACT</i>	if no other exception occurred

7.4.3.34 mod()

```
template<bool Q, bool R>
unsigned int half_float::detail::mod (
    unsigned int x,
    unsigned int y,
    int * quo = NULL)
```

Half precision positive modulus.

Template Parameters

<i>Q</i>	true to compute full quotient, false else
<i>R</i>	true to compute signed remainder, false for positive remainder

Parameters

<i>x</i>	first operand as positive finite half-precision value
<i>y</i>	second operand as positive finite half-precision value
<i>quo</i>	adress to store quotient at, nullptr if <i>Q</i> false

Returns

modulus of x / y

7.4.3.35 mulhi()

```
template<std::float_round_style R>
uint32 half_float::detail::mulhi (
    uint32 x,
    uint32 y)
```

upper part of 64-bit multiplication.

Template Parameters

<i>R</i>	rounding mode to use
----------	----------------------

Parameters

<i>x</i>	first factor
<i>y</i>	second factor

Returns

upper 32 bit of $x * y$

7.4.3.36 multiply64()

```
uint32 half_float::detail::multiply64 (
    uint32 x,
    uint32 y) [inline]
```

64-bit multiplication.

Parameters

<i>x</i>	first factor
<i>y</i>	second factor

Returns

upper 32 bit of $x * y$ rounded to nearest

7.4.3.37 overflow()

```
template<std::float_round_style R>
HALF_CONSTEXPR_NOERR unsigned int half_float::detail::overflow (
    unsigned int sign = 0)
```

Half-precision overflow.

Template Parameters

<i>R</i>	rounding mode to use
----------	----------------------

Parameters

<i>sign</i>	half-precision value with sign bit only
-------------	---

Returns

rounded overflowing half-precision value

Exceptions

<i>FE_OVERFLOW</i>	
--------------------	--

7.4.3.38 pole()

```
HALF_CONSTEXPR_NOERR unsigned int half_float::detail::pole (
    unsigned int sign = 0) [inline]
```

Raise pole error and return infinity.

Parameters

<i>sign</i>	half-precision value with sign bit only
-------------	---

Returns

half-precision infinity with sign of *sign*

Exceptions

<i>FE_DIVBYZERO</i>	
---------------------	--

7.4.3.39 raise()

```
void half_float::detail::raise (
    int  HALF_UNUSED_NOERRflags,
    bool HALF_UNUSED_NOERRcond = true) [inline]
```

Raise floating-point exception.

Parameters

<i>flags</i>	exceptions to raise
<i>cond</i>	condition to raise exceptions for

7.4.3.40 rounded()

```
template<std::float_round_style R, bool I>
HALF_CONSTEXPR_NOERR unsigned int half_float::detail::rounded (
    unsigned int value,
    int g,
    int s)
```

Round half-precision number.

Template Parameters

<i>R</i>	rounding mode to use
<i>I</i>	<code>true</code> to always raise INEXACT exception, <code>false</code> to raise only for rounded results

Parameters

<i>value</i>	finite half-precision number to round
<i>g</i>	guard bit (most significant discarded bit)
<i>s</i>	sticky bit (or of all but the most significant discarded bits)

Returns

rounded half-precision value

Exceptions

<i>FE_OVERFLOW</i>	on overflows
<i>FE_UNDERFLOW</i>	on underflows
<i>FE_INEXACT</i>	if value had to be rounded or <i>l</i> is <code>`true`</code>

7.4.3.41 select()

```
HALF_CONSTEXPR_NOERR unsigned int half_float::detail::select (
    unsigned int x,
    unsigned int HALF_UNUSED_NOERRy) [inline]
```

Select value or signaling NaN.

Parameters

<i>x</i>	preferred half-precision value
<i>y</i>	ignored half-precision value except for signaling NaN

Returns

y if signaling NaN, *x* otherwise

Exceptions

<i>FE_INVALID</i>	if <i>y</i> is signaling NaN
-------------------	------------------------------

7.4.3.42 sign_mask()

```
uint32 half_float::detail::sign_mask (
    uint32 arg) [inline]
```

Platform-independent sign mask.

Parameters

<i>arg</i>	integer value in two's complement
------------	-----------------------------------

Return values

<i>-1</i>	if <i>arg</i> negative
<i>0</i>	if <i>arg</i> positive

7.4.3.43 signal() [1/3]

```
HALF_CONSTEXPR_NOERR unsigned int half_float::detail::signal (
    unsigned int nan) [inline]
```

Signal and silence signaling NaN.

Parameters

<i>nan</i>	half-precision NaN value
------------	--------------------------

Returns

quiet NaN

Exceptions

<i>FE_INVALID</i>	if <i>nan</i> is signaling NaN
-------------------	--------------------------------

7.4.3.44 signal() [2/3]

```
HALF_CONSTEXPR_NOERR unsigned int half_float::detail::signal (
    unsigned int x,
    unsigned int y) [inline]
```

Signal and silence signaling NaNs.

Parameters

<i>x</i>	first half-precision value to check
<i>y</i>	second half-precision value to check

Returns

quiet NaN

Exceptions

<i>FE_INVALID</i>	if <i>x</i> or <i>y</i> is signaling NaN
-------------------	--

7.4.3.45 signal() [3/3]

```
HALF_CONSTEXPR_NOERR unsigned int half_float::detail::signal (
    unsigned int x,
    unsigned int y,
    unsigned int z) [inline]
```

Signal and silence signaling NaNs.

Parameters

<i>x</i>	first half-precision value to check
<i>y</i>	second half-precision value to check
<i>z</i>	third half-precision value to check

Returns

quiet NaN

Exceptions

<i>FE_INVALID</i>	if <i>x</i> , <i>y</i> or <i>z</i> is signaling NaN
-------------------	---

7.4.3.46 sincos()

```
std::pair< uint32, uint32 > half_float::detail::sincos (
    uint32 mz,
    unsigned int n = 31) [inline]
```

Fixed point sine and cosine. This uses the CORDIC algorithm in rotation mode.

Parameters

<i>mz</i>	angle in $[-\pi/2, \pi/2]$ as Q1.30
<i>n</i>	number of iterations (at most 31)

Returns

sine and cosine of *mz* as Q1.30

7.4.3.47 sqrt()

```
template<unsigned int F>
uint32 half_float::detail::sqrt (
    uint32 & r,
    int & exp)
```

Fixed point square root.

Template Parameters

<i>F</i>	number of fractional bits
----------	---------------------------

Parameters

<i>r</i>	radicand in Q1.F fixed point format
<i>exp</i>	exponent

Returns

square root as Q1.F/2

7.4.3.48 tangent_post()

```
template<std::float_round_style R>
unsigned int half_float::detail::tangent_post (
    uint32 my,
    uint32 mx,
    int exp,
    unsigned int sign = 0)
```

Division and postprocessing for tangents.

Template Parameters

<i>R</i>	rounding mode to use
----------	----------------------

Parameters

<i>my</i>	dividend as Q1.31
<i>mx</i>	divisor as Q1.31
<i>exp</i>	biased exponent of result
<i>sign</i>	sign bit of result

Returns

quotient converted to half-precision

Exceptions

<i>FE_OVERFLOW</i>	on overflows
<i>FE_UNDERFLOW</i>	on underflows
<i>FE_INEXACT</i>	if no other exception occurred

7.4.3.49 underflow()

```
template<std::float_round_style R>
HALF_CONSTEXPR_NOERR unsigned int half_float::detail::underflow (
    unsigned int sign = 0)
```

Half-precision underflow.

Template Parameters

<i>R</i>	rounding mode to use
----------	----------------------

Parameters

<i>sign</i>	half-precision value with sign bit only
-------------	---

Returns

rounded underflowing half-precision value

Exceptions

<i>FE_UNDERFLOW</i>	
---------------------	--

7.5 half.hpp

[Go to the documentation of this file.](#)

```

00001 // half - IEEE 754-based half-precision floating-point library.
00002 //
00003 // Copyright (c) 2012-2021 Christian Rau <rau@users.sourceforge.net>
00004 //
00005 // Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
    associated documentation
00006 // files (the "Software"), to deal in the Software without restriction, including without limitation
    the rights to use, copy,
00007 // modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit
    persons to whom the
00008 // Software is furnished to do so, subject to the following conditions:
00009 //
00010 // The above copyright notice and this permission notice shall be included in all copies or
    substantial portions of the Software.
00011 //
00012 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
    NOT LIMITED TO THE
00013 // WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
    SHALL THE AUTHORS OR
00014 // COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF
    CONTRACT, TORT OR OTHERWISE,
00015 // ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
    SOFTWARE.
00016
00017 // Version 2.2.0
00018
00021
00022 #ifndef HALF_HALF_HPP
00023 #define HALF_HALF_HPP
00024
00025 #define HALF_GCC_VERSION (__GNUC__*100+__GNUC_MINOR__)
00026
00027 #if defined(__INTEL_COMPILER)
00028     #define HALF_ICC_VERSION __INTEL_COMPILER
00029 #elif defined(__ICC)
00030     #define HALF_ICC_VERSION __ICC
00031 #elif defined(__ICL)
00032     #define HALF_ICC_VERSION __ICL
00033 #else
00034     #define HALF_ICC_VERSION 0
00035 #endif
00036
00037 // check C++11 language features
00038 #if defined(__clang__) // clang
00039     #if __has_feature(cxx_static_assert) && !defined(HALF_ENABLE_CPP11_STATIC_ASSERT)
00040         #define HALF_ENABLE_CPP11_STATIC_ASSERT 1
00041     #endif
00042     #if __has_feature(cxx_constexpr) && !defined(HALF_ENABLE_CPP11_CONSTEXPR)
00043         #define HALF_ENABLE_CPP11_CONSTEXPR 1
00044     #endif
00045     #if __has_feature(cxx_noexcept) && !defined(HALF_ENABLE_CPP11_NOEXCEPT)
00046         #define HALF_ENABLE_CPP11_NOEXCEPT 1
00047     #endif
00048     #if __has_feature(cxx_user_literals) && !defined(HALF_ENABLE_CPP11_USER_LITERALS)
00049         #define HALF_ENABLE_CPP11_USER_LITERALS 1
00050     #endif
00051     #if __has_feature(cxx_thread_local) && !defined(HALF_ENABLE_CPP11_THREAD_LOCAL)
00052         #define HALF_ENABLE_CPP11_THREAD_LOCAL 1
00053     #endif
00054     #if (defined(__GXX_EXPERIMENTAL_CXX0X__) || __cplusplus >= 201103L) &&
        !defined(HALF_ENABLE_CPP11_LONG_LONG)
00055         #define HALF_ENABLE_CPP11_LONG_LONG 1
00056     #endif
00057 #elif HALF_ICC_VERSION && defined(__INTEL_CXX11_MODE__) // Intel C++
00058     #if HALF_ICC_VERSION >= 1500 && !defined(HALF_ENABLE_CPP11_THREAD_LOCAL)
00059         #define HALF_ENABLE_CPP11_THREAD_LOCAL 1
00060     #endif
00061     #if HALF_ICC_VERSION >= 1500 && !defined(HALF_ENABLE_CPP11_USER_LITERALS)
00062         #define HALF_ENABLE_CPP11_USER_LITERALS 1
00063     #endif
00064     #if HALF_ICC_VERSION >= 1400 && !defined(HALF_ENABLE_CPP11_CONSTEXPR)
00065         #define HALF_ENABLE_CPP11_CONSTEXPR 1
00066     #endif
00067     #if HALF_ICC_VERSION >= 1400 && !defined(HALF_ENABLE_CPP11_NOEXCEPT)
00068         #define HALF_ENABLE_CPP11_NOEXCEPT 1
00069     #endif
00070     #if HALF_ICC_VERSION >= 1110 && !defined(HALF_ENABLE_CPP11_STATIC_ASSERT)
00071         #define HALF_ENABLE_CPP11_STATIC_ASSERT 1
00072     #endif
00073     #if HALF_ICC_VERSION >= 1110 && !defined(HALF_ENABLE_CPP11_LONG_LONG)
00074         #define HALF_ENABLE_CPP11_LONG_LONG 1
00075     #endif

```

```

00076 #elif defined(__GNUC__) // gcc
00077 #if defined(__GXX_EXPERIMENTAL_CXX0X__) || __cplusplus >= 201103L
00078 #if HALF_GCC_VERSION >= 408 && !defined(HALF_ENABLE_CPP11_THREAD_LOCAL)
00079 #define HALF_ENABLE_CPP11_THREAD_LOCAL 1
00080 #endif
00081 #if HALF_GCC_VERSION >= 407 && !defined(HALF_ENABLE_CPP11_USER_LITERALS)
00082 #define HALF_ENABLE_CPP11_USER_LITERALS 1
00083 #endif
00084 #if HALF_GCC_VERSION >= 406 && !defined(HALF_ENABLE_CPP11_CONSTEXPR)
00085 #define HALF_ENABLE_CPP11_CONSTEXPR 1
00086 #endif
00087 #if HALF_GCC_VERSION >= 406 && !defined(HALF_ENABLE_CPP11_NOEXCEPT)
00088 #define HALF_ENABLE_CPP11_NOEXCEPT 1
00089 #endif
00090 #if HALF_GCC_VERSION >= 403 && !defined(HALF_ENABLE_CPP11_STATIC_ASSERT)
00091 #define HALF_ENABLE_CPP11_STATIC_ASSERT 1
00092 #endif
00093 #if !defined(HALF_ENABLE_CPP11_LONG_LONG)
00094 #define HALF_ENABLE_CPP11_LONG_LONG 1
00095 #endif
00096 #endif
00097 #define HALF_TWOS_COMPLEMENT_INT 1
00098 #elif defined(_MSC_VER) // Visual C++
00099 #if _MSC_VER >= 1900 && !defined(HALF_ENABLE_CPP11_THREAD_LOCAL)
00100 #define HALF_ENABLE_CPP11_THREAD_LOCAL 1
00101 #endif
00102 #if _MSC_VER >= 1900 && !defined(HALF_ENABLE_CPP11_USER_LITERALS)
00103 #define HALF_ENABLE_CPP11_USER_LITERALS 1
00104 #endif
00105 #if _MSC_VER >= 1900 && !defined(HALF_ENABLE_CPP11_CONSTEXPR)
00106 #define HALF_ENABLE_CPP11_CONSTEXPR 1
00107 #endif
00108 #if _MSC_VER >= 1900 && !defined(HALF_ENABLE_CPP11_NOEXCEPT)
00109 #define HALF_ENABLE_CPP11_NOEXCEPT 1
00110 #endif
00111 #if _MSC_VER >= 1600 && !defined(HALF_ENABLE_CPP11_STATIC_ASSERT)
00112 #define HALF_ENABLE_CPP11_STATIC_ASSERT 1
00113 #endif
00114 #if _MSC_VER >= 1310 && !defined(HALF_ENABLE_CPP11_LONG_LONG)
00115 #define HALF_ENABLE_CPP11_LONG_LONG 1
00116 #endif
00117 #define HALF_TWOS_COMPLEMENT_INT 1
00118 #define HALF_POP_WARNINGS 1
00119 #pragma warning(push)
00120 #pragma warning(disable : 4099 4127 4146) //struct vs class, constant in if, negative unsigned
00121 #endif
00122
00123 // check C++11 library features
00124 #include <utility>
00125 #if defined(_LIBCPP_VERSION) // libc++
00126 #if defined(__GXX_EXPERIMENTAL_CXX0X__) || __cplusplus >= 201103
00127 #ifndef HALF_ENABLE_CPP11_TYPE_TRAITS
00128 #define HALF_ENABLE_CPP11_TYPE_TRAITS 1
00129 #endif
00130 #ifndef HALF_ENABLE_CPP11_CSTDINT
00131 #define HALF_ENABLE_CPP11_CSTDINT 1
00132 #endif
00133 #ifndef HALF_ENABLE_CPP11_CMATH
00134 #define HALF_ENABLE_CPP11_CMATH 1
00135 #endif
00136 #ifndef HALF_ENABLE_CPP11_HASH
00137 #define HALF_ENABLE_CPP11_HASH 1
00138 #endif
00139 #ifndef HALF_ENABLE_CPP11_CFENV
00140 #define HALF_ENABLE_CPP11_CFENV 1
00141 #endif
00142 #endif
00143 #elif defined(__GLIBCXX__) // libstdc++
00144 #if defined(__GXX_EXPERIMENTAL_CXX0X__) || __cplusplus >= 201103
00145 #ifdef __clang__
00146 #if __GLIBCXX__ >= 20080606 && !defined(HALF_ENABLE_CPP11_TYPE_TRAITS)
00147 #define HALF_ENABLE_CPP11_TYPE_TRAITS 1
00148 #endif
00149 #if __GLIBCXX__ >= 20080606 && !defined(HALF_ENABLE_CPP11_CSTDINT)
00150 #define HALF_ENABLE_CPP11_CSTDINT 1
00151 #endif
00152 #if __GLIBCXX__ >= 20080606 && !defined(HALF_ENABLE_CPP11_CMATH)
00153 #define HALF_ENABLE_CPP11_CMATH 1
00154 #endif
00155 #if __GLIBCXX__ >= 20080606 && !defined(HALF_ENABLE_CPP11_HASH)
00156 #define HALF_ENABLE_CPP11_HASH 1
00157 #endif
00158 #if __GLIBCXX__ >= 20080606 && !defined(HALF_ENABLE_CPP11_CFENV)
00159 #define HALF_ENABLE_CPP11_CFENV 1
00160 #endif
00161 #else
00162 #if HALF_GCC_VERSION >= 403 && !defined(HALF_ENABLE_CPP11_TYPE_TRAITS)

```

```

00163         #define HALF_ENABLE_CPP11_TYPE_TRAITS 1
00164     #endif
00165     #if HALF_GCC_VERSION >= 403 && !defined(HALF_ENABLE_CPP11_CSTDINT)
00166         #define HALF_ENABLE_CPP11_CSTDINT 1
00167     #endif
00168     #if HALF_GCC_VERSION >= 403 && !defined(HALF_ENABLE_CPP11_CMATH)
00169         #define HALF_ENABLE_CPP11_CMATH 1
00170     #endif
00171     #if HALF_GCC_VERSION >= 403 && !defined(HALF_ENABLE_CPP11_HASH)
00172         #define HALF_ENABLE_CPP11_HASH 1
00173     #endif
00174     #if HALF_GCC_VERSION >= 403 && !defined(HALF_ENABLE_CPP11_CFENV)
00175         #define HALF_ENABLE_CPP11_CFENV 1
00176     #endif
00177 #endif
00178 #endif
00179 #elif defined(_CPPLIB_VER) // Dinkumware/Visual C++
00180     #if _CPPLIB_VER >= 520 && !defined(HALF_ENABLE_CPP11_TYPE_TRAITS)
00181         #define HALF_ENABLE_CPP11_TYPE_TRAITS 1
00182     #endif
00183     #if _CPPLIB_VER >= 520 && !defined(HALF_ENABLE_CPP11_CSTDINT)
00184         #define HALF_ENABLE_CPP11_CSTDINT 1
00185     #endif
00186     #if _CPPLIB_VER >= 520 && !defined(HALF_ENABLE_CPP11_HASH)
00187         #define HALF_ENABLE_CPP11_HASH 1
00188     #endif
00189     #if _CPPLIB_VER >= 610 && !defined(HALF_ENABLE_CPP11_CMATH)
00190         #define HALF_ENABLE_CPP11_CMATH 1
00191     #endif
00192     #if _CPPLIB_VER >= 610 && !defined(HALF_ENABLE_CPP11_CFENV)
00193         #define HALF_ENABLE_CPP11_CFENV 1
00194     #endif
00195 #endif
00196 #undef HALF_GCC_VERSION
00197 #undef HALF_ICC_VERSION
00198
00199 // any error throwing C++ exceptions?
00200 #if defined(HALF_ERRHANDLING_THROW_INVALID) || defined(HALF_ERRHANDLING_THROW_DIVBYZERO) ||
    defined(HALF_ERRHANDLING_THROW_OVERFLOW) || defined(HALF_ERRHANDLING_THROW_UNDERFLOW) ||
    defined(HALF_ERRHANDLING_THROW_INEXACT)
00201 #define HALF_ERRHANDLING_THROWS 1
00202 #endif
00203
00204 // any error handling enabled?
00205 #define HALF_ERRHANDLING
    (HALF_ERRHANDLING_FLAGS || HALF_ERRHANDLING_ERRNO || HALF_ERRHANDLING_FENV || HALF_ERRHANDLING_THROWS)
00206
00207 #if HALF_ERRHANDLING
00208     #define HALF_UNUSED_NOERR(name) name
00209 #else
00210     #define HALF_UNUSED_NOERR(name)
00211 #endif
00212
00213 // support constexpr
00214 #if HALF_ENABLE_CPP11_CONSTEXPR
00215     #define HALF_CONSTEXPR constexpr
00216     #define HALF_CONSTEXPR_CONST constexpr
00217     #if HALF_ERRHANDLING
00218         #define HALF_CONSTEXPR_NOERR
00219     #else
00220         #define HALF_CONSTEXPR_NOERR constexpr
00221     #endif
00222 #else
00223     #define HALF_CONSTEXPR
00224     #define HALF_CONSTEXPR_CONST const
00225     #define HALF_CONSTEXPR_NOERR
00226 #endif
00227
00228 // support noexcept
00229 #if HALF_ENABLE_CPP11_NOEXCEPT
00230     #define HALF_NOEXCEPT noexcept
00231     #define HALF_NOTHROW noexcept
00232 #else
00233     #define HALF_NOEXCEPT
00234     #define HALF_NOTHROW throw()
00235 #endif
00236
00237 // support thread storage
00238 #if HALF_ENABLE_CPP11_THREAD_LOCAL
00239     #define HALF_THREAD_LOCAL thread_local
00240 #else
00241     #define HALF_THREAD_LOCAL static
00242 #endif
00243
00244 #include <utility>
00245 #include <algorithm>
00246 #include <istream>

```



```

00247 #include <ostream>
00248 #include <limits>
00249 #include <stdexcept>
00250 #include <climits>
00251 #include <cmath>
00252 #include <cstring>
00253 #include <cstdlib>
00254 #if HALF_ENABLE_CPP11_TYPE_TRAITS
00255     #include <type_traits>
00256 #endif
00257 #if HALF_ENABLE_CPP11_CSTDINT
00258     #include <cstdint>
00259 #endif
00260 #if HALF_ERRHANDLING_ERRNO
00261     #include <cerrno>
00262 #endif
00263 #if HALF_ENABLE_CPP11_CFENV
00264     #include <cfenv>
00265 #endif
00266 #if HALF_ENABLE_CPP11_HASH
00267     #include <functional>
00268 #endif
00269
00270
00271 #ifndef HALF_ENABLE_F16C_INTRINSICS
00272     #define HALF_ENABLE_F16C_INTRINSICS __F16C__
00273 #endif
00274 #if HALF_ENABLE_F16C_INTRINSICS
00275     #include <immintrin.h>
00276 #endif
00277
00278 #ifdef HALF_DOXYGEN_ONLY
00279 #define HALF_ARITHMETIC_TYPE (undefined)
00280
00281 #define HALF_ERRHANDLING_FLAGS 0
00282
00283 #define HALF_ERRHANDLING_ERRNO 0
00284
00285 #define HALF_ERRHANDLING_FENV 0
00286
00287 #define HALF_ERRHANDLING_THROW_INVALID (undefined)
00288
00289 #define HALF_ERRHANDLING_THROW_DIVBYZERO (undefined)
00290
00291 #define HALF_ERRHANDLING_THROW_OVERFLOW (undefined)
00292
00293 #define HALF_ERRHANDLING_THROW_UNDERFLOW (undefined)
00294
00295 #define HALF_ERRHANDLING_THROW_INEXACT (undefined)
00296 #endif
00297
00298 #ifndef HALF_ERRHANDLING_OVERFLOW_TO_INEXACT
00299 #define HALF_ERRHANDLING_OVERFLOW_TO_INEXACT 1
00300 #endif
00301
00302 #ifndef HALF_ERRHANDLING_UNDERFLOW_TO_INEXACT
00303 #define HALF_ERRHANDLING_UNDERFLOW_TO_INEXACT 1
00304 #endif
00305
00306 #ifndef HALF_ROUND_STYLE
00307     #define HALF_ROUND_STYLE 1 // = std::round_to_nearest
00308 #endif
00309
00310 #define HUGE_VALH std::numeric_limits<half_float::half>::infinity()
00311
00312 #define FP_FAST_FMAH 1
00313
00314 #define HLF_ROUNDS HALF_ROUND_STYLE
00315
00316 #ifndef FP_ILOGB0
00317     #define FP_ILOGB0 INT_MIN
00318 #endif
00319
00320 #ifndef FP_ILOGBNAN
00321     #define FP_ILOGBNAN INT_MAX
00322 #endif
00323
00324 #ifndef FP_SUBNORMAL
00325     #define FP_SUBNORMAL 0
00326 #endif
00327
00328 #ifndef FP_ZERO
00329     #define FP_ZERO 1
00330 #endif
00331
00332 #ifndef FP_NAN
00333     #define FP_NAN 2
00334 #endif
00335
00336 #ifndef FP_INFINITE
00337     #define FP_INFINITE 3
00338 #endif
00339

```

```

00416 #ifndef FP_NORMAL
00417     #define FP_NORMAL      4
00418 #endif
00419
00420 #if !HALF_ENABLE_CPP11_CFENV && !defined(FE_ALL_EXCEPT)
00421     #define FE_INVALID      0x10
00422     #define FE_DIVBYZERO    0x08
00423     #define FE_OVERFLOW     0x04
00424     #define FE_UNDERFLOW   0x02
00425     #define FE_INEXACT      0x01
00426     #define FE_ALL_EXCEPT (FE_INVALID|FE_DIVBYZERO|FE_OVERFLOW|FE_UNDERFLOW|FE_INEXACT)
00427 #endif
00428
00429
00432 namespace half_float
00433 {
00434     class half;
00435
00436 #if HALF_ENABLE_CPP11_USER_LITERALS
00443     namespace literal
00444     {
00445         half operator "" _h(long double);
00446     }
00447 #endif
00448
00451     namespace detail
00452     {
00453         #if HALF_ENABLE_CPP11_TYPE_TRAITS
00455             template<bool B,typename T,typename F> struct conditional : std::conditional<B,T,F> {};
00456
00458             template<bool B> struct bool_type : std::integral_constant<bool,B> {};
00459             using std::true_type;
00460             using std::false_type;
00461
00463             template<typename T> struct is_float : std::is_floating_point<T> {};
00464         #else
00466             template<bool,typename T,typename> struct conditional { typedef T type; };
00467             template<typename T,typename F> struct conditional<false,T,F> { typedef F type; };
00468
00470             template<bool> struct bool_type {};
00471             typedef bool_type<true> true_type;
00472             typedef bool_type<false> false_type;
00473
00475             template<typename> struct is_float : false_type {};
00476             template<typename T> struct is_float<const T> : is_float<T> {};
00477             template<typename T> struct is_float<volatile T> : is_float<T> {};
00478             template<typename T> struct is_float<const volatile T> : is_float<T> {};
00479             template<> struct is_float<float> : true_type {};
00480             template<> struct is_float<double> : true_type {};
00481             template<> struct is_float<long double> : true_type {};
00482         #endif
00483
00485             template<typename T> struct bits { typedef unsigned char type; };
00486             template<typename T> struct bits<const T> : bits<T> {};
00487             template<typename T> struct bits<volatile T> : bits<T> {};
00488             template<typename T> struct bits<const volatile T> : bits<T> {};
00489
00490         #if HALF_ENABLE_CPP11_CSTDINT
00492             typedef std::uint_least16_t uint16;
00493
00495             typedef std::uint_fast32_t uint32;
00496
00498             typedef std::int_fast32_t int32;
00499
00501             template<> struct bits<float> { typedef std::uint_least32_t type; };
00502
00504             template<> struct bits<double> { typedef std::uint_least64_t type; };
00505         #else
00507             typedef unsigned short uint16;
00508
00510             typedef unsigned long uint32;
00511
00513             typedef long int32;
00514
00516             template<> struct bits<float> : conditional<std::numeric_limits<unsigned
int>::digits>=32,unsigned int,unsigned long> {};
00517
00518             #if HALF_ENABLE_CPP11_LONG_LONG
00520                 template<> struct bits<double> : conditional<std::numeric_limits<unsigned
long>::digits>=64,unsigned long,unsigned long long> {};
00521             #else
00523                 template<> struct bits<double> { typedef unsigned long type; };
00524             #endif
00525         #endif
00526
00527         #ifdef HALF_ARITHMETIC_TYPE
00529             typedef HALF_ARITHMETIC_TYPE internal_t;

```

```

00530     #endif
00531
00532     struct binary_t {};
00533
00534     HALF_CONSTEXPR_CONST binary_t binary = binary_t();
00535
00536     template<typename T> bool builtin_isinf(T arg)
00537     {
00538         #if HALF_ENABLE_CPP11_CMATH
00539             return std::isinf(arg);
00540         #elif defined(_MSC_VER)
00541             return !::_finite(static_cast<double>(arg)) && !::_isnan(static_cast<double>(arg));
00542         #else
00543             return arg == std::numeric_limits<T>::infinity() || arg ==
00544             -std::numeric_limits<T>::infinity();
00545         #endif
00546     }
00547
00548     template<typename T> bool builtin_isnan(T arg)
00549     {
00550         #if HALF_ENABLE_CPP11_CMATH
00551             return std::isnan(arg);
00552         #elif defined(_MSC_VER)
00553             return !::_isnan(static_cast<double>(arg)) != 0;
00554         #else
00555             return arg != arg;
00556         #endif
00557     }
00558
00559     template<typename T> bool builtin_signbit(T arg)
00560     {
00561         #if HALF_ENABLE_CPP11_CMATH
00562             return std::signbit(arg);
00563         #else
00564             return arg < T() || (arg == T() && T(1)/arg < T());
00565         #endif
00566     }
00567
00568     inline uint32 sign_mask(uint32 arg)
00569     {
00570         static const int N = std::numeric_limits<uint32>::digits - 1;
00571         #if HALF_TWOS_COMPLEMENT_INT
00572             return static_cast<int32>(arg) >> N;
00573         #else
00574             return -(arg >> N) & 1;
00575         #endif
00576     }
00577
00578     inline uint32 arithmetic_shift(uint32 arg, int i)
00579     {
00580         #if HALF_TWOS_COMPLEMENT_INT
00581             return static_cast<int32>(arg) >> i;
00582         #else
00583             return static_cast<int32>(arg) / (static_cast<int32>(1) << i) -
00584             ((arg >> (std::numeric_limits<uint32>::digits-1)) & 1);
00585         #endif
00586     }
00587
00588     inline int& errflags() { HALF_THREAD_LOCAL int flags = 0; return flags; }
00589
00590     inline void raise(int HALF_UNUSED_NOERR(flags), bool HALF_UNUSED_NOERR(cond) = true)
00591     {
00592         #if HALF_ERRHANDLING
00593             if(!cond)
00594                 return;
00595         #if HALF_ERRHANDLING_FLAGS
00596             errflags() |= flags;
00597         #endif
00598         #if HALF_ERRHANDLING_ERRNO
00599             if(flags & FE_INVALID)
00600                 errno = EDOM;
00601             else if(flags & (FE_DIVBYZERO|FE_OVERFLOW|FE_UNDERFLOW))
00602                 errno = ERANGE;
00603         #endif
00604         #if HALF_ERRHANDLING_FENV && HALF_ENABLE_CPP11_CFENV
00605             std::fraiseexcept(flags);
00606         #endif
00607         #ifdef HALF_ERRHANDLING_THROW_INVALID
00608             if(flags & FE_INVALID)
00609                 throw std::domain_error(HALF_ERRHANDLING_THROW_INVALID);
00610         #endif
00611         #ifdef HALF_ERRHANDLING_THROW_DIVBYZERO
00612             if(flags & FE_DIVBYZERO)
00613                 throw std::domain_error(HALF_ERRHANDLING_THROW_DIVBYZERO);
00614         #endif
00615     }

```

```

00650     #ifdef HALF_ERRHANDLING_THROW_OVERFLOW
00651         if (flags & FE_OVERFLOW)
00652             throw std::overflow_error(HALF_ERRHANDLING_THROW_OVERFLOW);
00653     #endif
00654     #ifdef HALF_ERRHANDLING_THROW_UNDERFLOW
00655         if (flags & FE_UNDERFLOW)
00656             throw std::underflow_error(HALF_ERRHANDLING_THROW_UNDERFLOW);
00657     #endif
00658     #ifdef HALF_ERRHANDLING_THROW_INEXACT
00659         if (flags & FE_INEXACT)
00660             throw std::range_error(HALF_ERRHANDLING_THROW_INEXACT);
00661     #endif
00662     #if HALF_ERRHANDLING_UNDERFLOW_TO_INEXACT
00663         if ((flags & FE_UNDERFLOW) && !(flags & FE_INEXACT))
00664             raise(FE_INEXACT);
00665     #endif
00666     #if HALF_ERRHANDLING_OVERFLOW_TO_INEXACT
00667         if ((flags & FE_OVERFLOW) && !(flags & FE_INEXACT))
00668             raise(FE_INEXACT);
00669     #endif
00670     #endif
00671 }
00672
00673 inline HALF_CONSTEXPR_NOERR bool compsignal(unsigned int x, unsigned int y)
00674 {
00675     #if HALF_ERRHANDLING
00676         raise(FE_INVALID, (x&0x7FFF)>0x7C00 || (y&0x7FFF)>0x7C00);
00677     #endif
00678     return (x&0x7FFF) > 0x7C00 || (y&0x7FFF) > 0x7C00;
00679 }
00680
00681 inline HALF_CONSTEXPR_NOERR unsigned int signal(unsigned int nan)
00682 {
00683     #if HALF_ERRHANDLING
00684         raise(FE_INVALID, !(nan&0x200));
00685     #endif
00686     return nan | 0x200;
00687 }
00688
00689 inline HALF_CONSTEXPR_NOERR unsigned int signal(unsigned int x, unsigned int y)
00690 {
00691     #if HALF_ERRHANDLING
00692         raise(FE_INVALID, ((x&0x7FFF)>0x7C00 && !(x&0x200)) || ((y&0x7FFF)>0x7C00 && !(y&0x200)));
00693     #endif
00694     return ((x&0x7FFF)>0x7C00) ? (x|0x200) : (y|0x200);
00695 }
00696
00697 inline HALF_CONSTEXPR_NOERR unsigned int signal(unsigned int x, unsigned int y, unsigned int
00698 z)
00699 {
00700     #if HALF_ERRHANDLING
00701         raise(FE_INVALID, ((x&0x7FFF)>0x7C00 && !(x&0x200)) || ((y&0x7FFF)>0x7C00 && !(y&0x200))
00702 || ((z&0x7FFF)>0x7C00 && !(z&0x200)));
00703     #endif
00704     return ((x&0x7FFF)>0x7C00) ? (x|0x200) : ((y&0x7FFF)>0x7C00) ? (y|0x200) : (z|0x200);
00705 }
00706
00707 inline HALF_CONSTEXPR_NOERR unsigned int select(unsigned int x, unsigned int
00708 HALF_UNUSED_NOERR(y))
00709 {
00710     #if HALF_ERRHANDLING
00711         return ((y&0x7FFF)>0x7C00) && !(y&0x200) ? signal(y) : x;
00712     #else
00713         return x;
00714     #endif
00715 }
00716
00717 inline HALF_CONSTEXPR_NOERR unsigned int invalid()
00718 {
00719     #if HALF_ERRHANDLING
00720         raise(FE_INVALID);
00721     #endif
00722     return 0x7FFF;
00723 }
00724
00725 inline HALF_CONSTEXPR_NOERR unsigned int pole(unsigned int sign = 0)
00726 {
00727     #if HALF_ERRHANDLING
00728         raise(FE_DIVBYZERO);
00729     #endif
00730     return sign | 0x7C00;
00731 }
00732
00733 inline HALF_CONSTEXPR_NOERR unsigned int check_underflow(unsigned int arg)
00734 {
00735     #if HALF_ERRHANDLING && !HALF_ERRHANDLING_UNDERFLOW_TO_INEXACT
00736         raise(FE_UNDERFLOW, !(arg&0x7C00));
00737     #endif

```

```

00771         #endif
00772         return arg;
00773     }
00774
00775
00776
00777
00778
00784     template<std::float_round_style R> HALF_CONSTEXPR_NOERR unsigned int overflow(unsigned int
sign = 0)
00785     {
00786         #if HALF_ERRHANDLING
00787             raise(FE_OVERFLOW);
00788         #endif
00789         return (R==std::round_toward_infinity) ? (sign+0x7C00-(sign>>15)) :
00790             (R==std::round_toward_neg_infinity) ? (sign+0x7BFF+(sign>>15)) :
00791             (R==std::round_toward_zero) ? (sign|0x7BFF) :
00792             (sign|0x7C00);
00793     }
00794
00795
00800     template<std::float_round_style R> HALF_CONSTEXPR_NOERR unsigned int underflow(unsigned int
sign = 0)
00801     {
00802         #if HALF_ERRHANDLING
00803             raise(FE_UNDERFLOW);
00804         #endif
00805         return (R==std::round_toward_infinity) ? (sign+1-(sign>>15)) :
00806             (R==std::round_toward_neg_infinity) ? (sign+(sign>>15)) :
00807             sign;
00808     }
00809
00820     template<std::float_round_style R,bool I> HALF_CONSTEXPR_NOERR unsigned int rounded(unsigned
int value, int g, int s)
00821     {
00822         #if HALF_ERRHANDLING
00823             value += (R==std::round_to_nearest) ? (g&(s|value)) :
00824                 (R==std::round_toward_infinity) ? (~value>>15)&(g|s) :
00825                 (R==std::round_toward_neg_infinity) ? ((value>>15)&(g|s)) : 0;
00826             if((value&0x7C00) == 0x7C00)
00827                 raise(FE_OVERFLOW);
00828             else if(value & 0x7C00)
00829                 raise(FE_INEXACT, I || (g|s)!=0);
00830             else
00831                 raise(FE_UNDERFLOW, !(HALF_ERRHANDLING_UNDERFLOW_TO_INEXACT) || I || (g|s)!=0);
00832             return value;
00833         #else
00834             return (R==std::round_to_nearest) ? (value+(g&(s|value))) :
00835                 (R==std::round_toward_infinity) ? (value+~(value>>15)&(g|s)) :
00836                 (R==std::round_toward_neg_infinity) ? (value+((value>>15)&(g|s))) :
00837                 value;
00838         #endif
00839     }
00840
00849     template<std::float_round_style R,bool E,bool I> unsigned int integral(unsigned int value)
00850     {
00851         unsigned int abs = value & 0x7FFF;
00852         if(abs < 0x3C00)
00853         {
00854             raise(FE_INEXACT, I);
00855             return ((R==std::round_to_nearest) ? (0x3C00&~static_cast<unsigned>(abs>=(0x3800+E)))
:
00856                 (R==std::round_toward_infinity) ? (0x3C00&~(value>>15)&(abs!=0))) :
00857                 (R==std::round_toward_neg_infinity) ?
(0x3C00&~static_cast<unsigned>(value>0x8000)) :
00858                 0) | (value&0x8000);
00859         }
00860         if(abs >= 0x6400)
00861             return (abs>0x7C00) ? signal(value) : value;
00862         unsigned int exp = 25 - (abs>>10), mask = (1<<exp) - 1;
00863         raise(FE_INEXACT, I && (value&mask));
00864         return (( (R==std::round_to_nearest) ? ((1<<(exp-1))-~(value>>exp)&E) :
00865             (R==std::round_toward_infinity) ? (mask&((value>>15)-1)) :
00866             (R==std::round_toward_neg_infinity) ? (mask&~(value>>15)) :
00867             0) + value) & ~mask;
00868     }
00869
00884     template<std::float_round_style R,unsigned int F,bool S,bool N,bool I> unsigned int
fixed2half(uint32 m, int exp = 14, unsigned int sign = 0, int s = 0)
00885     {
00886         if(S)
00887         {
00888             uint32 msign = sign_mask(m);
00889             m = (m^msign) - msign;
00890             sign = msign & 0x8000;
00891         }
00892         if(N)
00893             for(; m<(static_cast<uint32>(1)<<(F) && exp; m<=1,--exp) ;
00894         else if(exp < 0)
00895             return rounded<R,I>(sign+(m>>(F-10-exp)), (m>>(F-11-exp))&1,
s | (m&((static_cast<uint32>(1)<<(F-11-exp))-1))!=0));

```



```

00964     0xE800, 0xEC00, 0xF000, 0xF400, 0xF800, 0xFBFF,
        0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF,
00965     0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF,
        0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF,
00966     0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF,
        0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF,
00967     0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF,
        0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF,
00968     0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF,
        0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF,
00969     0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF,
        0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF,
00970     0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF,
        0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF,
00971     static const unsigned char shift_table[256] = {
00972         24, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25,
        25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25,
00973         25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25,
        25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25,
00974         25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25,
        25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25,
00975         25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25,
        13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
00976         13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
        24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24,
00977         24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24,
        24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24,
00978         24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24,
        24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24,
00979         24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24,
        24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24,
00980         int sexp = fbits > 23, exp = sexp & 0xFF, i = shift_table[exp];
00981         fbits &= 0x7FFFFF;
00982         uint32 m = (fbits | ((exp != 0) << 23)) & -static_cast<uint32>(exp != 0xFF);
00983         return rounded<R, false>(base_table[sexp] + (fbits > i), (m > (i - 1)) & 1,
        ((static_cast<uint32>(1) << (i - 1)) - 1) & m) != 0);
00984     #endif
00985     #endif
00986 }
00987
00988 template<std::float_round_style R> unsigned int float2half_impl(double value, true_type)
00989 {
00990     #if HALF_ENABLE_F16C_INTRINSICS
00991         if (R == std::round_indeterminate)
00992             return _mm_cvtsil28_si32(_mm_cvtps_ph(_mm_cvtpd_ps(_mm_set_sd(value))),
        _MM_FROUND_CUR_DIRECTION));
00993     #endif
00994     bits<double>::type dbits;
00995     std::memcpy(&dbits, &value, sizeof(double));
00996     uint32 hi = dbits >> 32, lo = dbits & 0xFFFFFFFF;
00997     unsigned int sign = (hi >> 16) & 0x8000;
00998     hi &= 0x7FFFFFFF;
00999     if (hi >= 0x7FF00000)
01000         return sign | 0x7C00 | ((dbits & 0xFFFFFFFFFFFFFFFF) ? (0x200 | ((hi >> 10) & 0x3FF)) : 0);
01001     if (hi >= 0x40F00000)
01002         return overflow<R>(sign);
01003     if (hi >= 0x3F100000)
01004         return rounded<R, false>(sign | ((hi >> 20) - 1008) << 10 | ((hi >> 10) & 0x3FF), (hi >> 9) & 1,
        ((hi & 0x1FF) | lo) != 0);
01005     if (hi >= 0x3E600000)
01006     {
01007         int i = 1018 - (hi >> 20);
01008         hi = (hi & 0xFFFFF) | 0x1000000;
01009         return rounded<R, false>(sign | (hi >> (i + 1)), (hi >> i) & 1,
        ((hi & ((static_cast<uint32>(1) << i) - 1)) | lo) != 0);
01010     }
01011     if ((hi | lo) != 0)
01012         return underflow<R>(sign);
01013     return sign;
01014 }
01015
01016 template<std::float_round_style R, typename T> unsigned int float2half_impl(T value, ...)
01017 {
01018     unsigned int hbits = static_cast<unsigned>(builtin_signbit(value)) << 15;
01019     if (value == T())
01020         return hbits;
01021     if (builtin_isnan(value))
01022         return hbits | 0x7FFF;
01023     if (builtin_isinf(value))
01024         return hbits | 0x7C00;
01025     int exp;
01026     std::frexp(value, &exp);
01027     if (exp > 16)
01028         return overflow<R>(hbits);
01029     if (exp < -13)
01030         value = std::ldexp(value, 25);

```

```

01046         else
01047         {
01048             value = std::ldexp(value, 12-exp);
01049             hbits |= ((exp+13)<<10);
01050         }
01051         T ival, frac = std::modf(value, &ival);
01052         int m = std::abs(static_cast<int>(ival));
01053         return rounded<R, false>(hbits+(m>1), m&1, frac!=T());
01054     }
01055
01064     template<std::float_round_style R, typename T> unsigned int float2half(T value)
01065     {
01066         return float2half_impl<R>(value,
bool_type<std::numeric_limits<T>::is_iec559&&sizeof(typename bits<T>::type)==sizeof(T)>());
01067     }
01068
01076     template<std::float_round_style R, typename T> unsigned int int2half(T value)
01077     {
01078         unsigned int bits = static_cast<unsigned>(value<0) << 15;
01079         if(!value)
01080             return bits;
01081         if(bits)
01082             value = -value;
01083         if(value > 0xFFFF)
01084             return overflow<R>(bits);
01085         unsigned int m = static_cast<unsigned int>(value), exp = 24;
01086         for(; m<0x400; m<=1,--exp) ;
01087         for(; m>0x7FF; m>=1,++exp) ;
01088         bits |= (exp<<10) + m;
01089         return (exp>24) ? rounded<R, false>(bits, (value>=0)&1, ((1<<(exp-25))-1)&value)!=0)
: bits;
01090     }
01091
01096     inline float half2float_impl(unsigned int value, float, true_type)
01097     {
01098         #if HALF_ENABLE_F16C_INTRINSICS
01099             return _mm_cvtss_f32(_mm_cvtph_ps(_mm_cvtsi32_si128(value)));
01100         #else
01101         #if 0
01102             bits<float>::type fbits = static_cast<bits<float>::type>(value<0x8000) << 16;
01103             int abs = value & 0x7FFF;
01104             if(abs)
01105             {
01106                 fbits |= 0x38000000 << static_cast<unsigned>(abs)>=0x7C00);
01107                 for(; abs<0x400; abs<=1, fbits-=0x800000) ;
01108                 fbits += static_cast<bits<float>::type>(abs) << 13;
01109             }
01110         #else
01111             static const bits<float>::type mantissa_table[2048] = {
01112                 0x00000000, 0x33800000, 0x34000000, 0x34400000, 0x34800000, 0x34A00000, 0x34C00000,
0x34E00000, 0x35000000, 0x35100000, 0x35200000, 0x35300000, 0x35400000, 0x35500000, 0x35600000,
0x35700000,
01113                 0x35800000, 0x35880000, 0x35900000, 0x35980000, 0x35A00000, 0x35A80000, 0x35B00000,
0x35B80000, 0x35C00000, 0x35C80000, 0x35D00000, 0x35D80000, 0x35E00000, 0x35E80000, 0x35F00000,
0x35F80000,
01114                 0x36000000, 0x36040000, 0x36080000, 0x360C0000, 0x36100000, 0x36140000, 0x36180000,
0x361C0000, 0x36200000, 0x36240000, 0x36280000, 0x362C0000, 0x36300000, 0x36340000, 0x36380000,
0x363C0000,
01115                 0x36400000, 0x36440000, 0x36480000, 0x364C0000, 0x36500000, 0x36540000, 0x36580000,
0x365C0000, 0x36600000, 0x36640000, 0x36680000, 0x366C0000, 0x36700000, 0x36740000, 0x36780000,
0x367C0000,
01116                 0x36800000, 0x36820000, 0x36840000, 0x36860000, 0x36880000, 0x368A0000, 0x368C0000,
0x368E0000, 0x36900000, 0x36920000, 0x36940000, 0x36960000, 0x36980000, 0x369A0000, 0x369C0000,
0x369E0000,
01117                 0x36A00000, 0x36A20000, 0x36A40000, 0x36A60000, 0x36A80000, 0x36AA0000, 0x36AC0000,
0x36AE0000, 0x36B00000, 0x36B20000, 0x36B40000, 0x36B60000, 0x36B80000, 0x36BA0000, 0x36BC0000,
0x36BE0000,
01118                 0x36C00000, 0x36C20000, 0x36C40000, 0x36C60000, 0x36C80000, 0x36CA0000, 0x36CC0000,
0x36CE0000, 0x36D00000, 0x36D20000, 0x36D40000, 0x36D60000, 0x36D80000, 0x36DA0000, 0x36DC0000,
0x36DE0000,
01119                 0x36E00000, 0x36E20000, 0x36E40000, 0x36E60000, 0x36E80000, 0x36EA0000, 0x36EC0000,
0x36EE0000, 0x36F00000, 0x36F20000, 0x36F40000, 0x36F60000, 0x36F80000, 0x36FA0000, 0x36FC0000,
0x36FE0000,
01120                 0x37000000, 0x37010000, 0x37020000, 0x37030000, 0x37040000, 0x37050000, 0x37060000,
0x37070000, 0x37080000, 0x37090000, 0x370A0000, 0x370B0000, 0x370C0000, 0x370D0000, 0x370E0000,
0x370F0000,
01121                 0x37100000, 0x37110000, 0x37120000, 0x37130000, 0x37140000, 0x37150000, 0x37160000,
0x37170000, 0x37180000, 0x37190000, 0x371A0000, 0x371B0000, 0x371C0000, 0x371D0000, 0x371E0000,
0x371F0000,
01122                 0x37200000, 0x37210000, 0x37220000, 0x37230000, 0x37240000, 0x37250000, 0x37260000,
0x37270000, 0x37280000, 0x37290000, 0x372A0000, 0x372B0000, 0x372C0000, 0x372D0000, 0x372E0000,
0x372F0000,
01123                 0x37300000, 0x37310000, 0x37320000, 0x37330000, 0x37340000, 0x37350000, 0x37360000,
0x37370000, 0x37380000, 0x37390000, 0x373A0000, 0x373B0000, 0x373C0000, 0x373D0000, 0x373E0000,
0x373F0000,
01124                 0x37400000, 0x37410000, 0x37420000, 0x37430000, 0x37440000, 0x37450000, 0x37460000,
0x37470000, 0x37480000, 0x37490000, 0x374A0000, 0x374B0000, 0x374C0000, 0x374D0000, 0x374E0000,

```



```
0x374F0000,
01125      0x37500000, 0x37510000, 0x37520000, 0x37530000, 0x37540000, 0x37550000, 0x37560000,
0x37570000, 0x37580000, 0x37590000, 0x375A0000, 0x375B0000, 0x375C0000, 0x375D0000, 0x375E0000,
0x375F0000,
01126      0x37600000, 0x37610000, 0x37620000, 0x37630000, 0x37640000, 0x37650000, 0x37660000,
0x37670000, 0x37680000, 0x37690000, 0x376A0000, 0x376B0000, 0x376C0000, 0x376D0000, 0x376E0000,
0x376F0000,
01127      0x37700000, 0x37710000, 0x37720000, 0x37730000, 0x37740000, 0x37750000, 0x37760000,
0x37770000, 0x37780000, 0x37790000, 0x377A0000, 0x377B0000, 0x377C0000, 0x377D0000, 0x377E0000,
0x377F0000,
01128      0x37800000, 0x37808000, 0x37810000, 0x37818000, 0x37820000, 0x37828000, 0x37830000,
0x37838000, 0x37840000, 0x37848000, 0x37850000, 0x37858000, 0x37860000, 0x37868000, 0x37870000,
0x37878000,
01129      0x37880000, 0x37888000, 0x37890000, 0x37898000, 0x378A0000, 0x378A8000, 0x378B0000,
0x378B8000, 0x378C0000, 0x378C8000, 0x378D0000, 0x378D8000, 0x378E0000, 0x378E8000, 0x378F0000,
0x378F8000,
01130      0x37900000, 0x37908000, 0x37910000, 0x37918000, 0x37920000, 0x37928000, 0x37930000,
0x37938000, 0x37940000, 0x37948000, 0x37950000, 0x37958000, 0x37960000, 0x37968000, 0x37970000,
0x37978000,
01131      0x37980000, 0x37988000, 0x37990000, 0x37998000, 0x379A0000, 0x379A8000, 0x379B0000,
0x379B8000, 0x379C0000, 0x379C8000, 0x379D0000, 0x379D8000, 0x379E0000, 0x379E8000, 0x379F0000,
0x379F8000,
01132      0x37A00000, 0x37A08000, 0x37A10000, 0x37A18000, 0x37A20000, 0x37A28000, 0x37A30000,
0x37A38000, 0x37A40000, 0x37A48000, 0x37A50000, 0x37A58000, 0x37A60000, 0x37A68000, 0x37A70000,
0x37A78000,
01133      0x37A80000, 0x37A88000, 0x37A90000, 0x37A98000, 0x37AA0000, 0x37AA8000, 0x37AB0000,
0x37AB8000, 0x37AC0000, 0x37AC8000, 0x37AD0000, 0x37AD8000, 0x37AE0000, 0x37AE8000, 0x37AF0000,
0x37AF8000,
01134      0x37B00000, 0x37B08000, 0x37B10000, 0x37B18000, 0x37B20000, 0x37B28000, 0x37B30000,
0x37B38000, 0x37B40000, 0x37B48000, 0x37B50000, 0x37B58000, 0x37B60000, 0x37B68000, 0x37B70000,
0x37B78000,
01135      0x37B80000, 0x37B88000, 0x37B90000, 0x37B98000, 0x37BA0000, 0x37BA8000, 0x37BB0000,
0x37BB8000, 0x37BC0000, 0x37BC8000, 0x37BD0000, 0x37BD8000, 0x37BE0000, 0x37BE8000, 0x37BF0000,
0x37BF8000,
01136      0x37C00000, 0x37C08000, 0x37C10000, 0x37C18000, 0x37C20000, 0x37C28000, 0x37C30000,
0x37C38000, 0x37C40000, 0x37C48000, 0x37C50000, 0x37C58000, 0x37C60000, 0x37C68000, 0x37C70000,
0x37C78000,
01137      0x37C80000, 0x37C88000, 0x37C90000, 0x37C98000, 0x37CA0000, 0x37CA8000, 0x37CB0000,
0x37CB8000, 0x37CC0000, 0x37CC8000, 0x37CD0000, 0x37CD8000, 0x37CE0000, 0x37CE8000, 0x37CF0000,
0x37CF8000,
01138      0x37D00000, 0x37D08000, 0x37D10000, 0x37D18000, 0x37D20000, 0x37D28000, 0x37D30000,
0x37D38000, 0x37D40000, 0x37D48000, 0x37D50000, 0x37D58000, 0x37D60000, 0x37D68000, 0x37D70000,
0x37D78000,
01139      0x37D80000, 0x37D88000, 0x37D90000, 0x37D98000, 0x37DA0000, 0x37DA8000, 0x37DB0000,
0x37DB8000, 0x37DC0000, 0x37DC8000, 0x37DD0000, 0x37DD8000, 0x37DE0000, 0x37DE8000, 0x37DF0000,
0x37DF8000,
01140      0x37E00000, 0x37E08000, 0x37E10000, 0x37E18000, 0x37E20000, 0x37E28000, 0x37E30000,
0x37E38000, 0x37E40000, 0x37E48000, 0x37E50000, 0x37E58000, 0x37E60000, 0x37E68000, 0x37E70000,
0x37E78000,
01141      0x37E80000, 0x37E88000, 0x37E90000, 0x37E98000, 0x37EA0000, 0x37EA8000, 0x37EB0000,
0x37EB8000, 0x37EC0000, 0x37EC8000, 0x37ED0000, 0x37ED8000, 0x37EE0000, 0x37EE8000, 0x37EF0000,
0x37EF8000,
01142      0x37F00000, 0x37F08000, 0x37F10000, 0x37F18000, 0x37F20000, 0x37F28000, 0x37F30000,
0x37F38000, 0x37F40000, 0x37F48000, 0x37F50000, 0x37F58000, 0x37F60000, 0x37F68000, 0x37F70000,
0x37F78000,
01143      0x37F80000, 0x37F88000, 0x37F90000, 0x37F98000, 0x37FA0000, 0x37FA8000, 0x37FB0000,
0x37FB8000, 0x37FC0000, 0x37FC8000, 0x37FD0000, 0x37FD8000, 0x37FE0000, 0x37FE8000, 0x37FF0000,
0x37FF8000,
01144      0x38000000, 0x38004000, 0x38008000, 0x3800C000, 0x38010000, 0x38014000, 0x38018000,
0x3801C000, 0x38020000, 0x38024000, 0x38028000, 0x3802C000, 0x38030000, 0x38034000, 0x38038000,
0x3803C000,
01145      0x38040000, 0x38044000, 0x38048000, 0x3804C000, 0x38050000, 0x38054000, 0x38058000,
0x3805C000, 0x38060000, 0x38064000, 0x38068000, 0x3806C000, 0x38070000, 0x38074000, 0x38078000,
0x3807C000,
01146      0x38080000, 0x38084000, 0x38088000, 0x3808C000, 0x38090000, 0x38094000, 0x38098000,
0x3809C000, 0x380A0000, 0x380A4000, 0x380A8000, 0x380AC000, 0x380B0000, 0x380B4000, 0x380B8000,
0x380BC000,
01147      0x380C0000, 0x380C4000, 0x380C8000, 0x380CC000, 0x380D0000, 0x380D4000, 0x380D8000,
0x380DC000, 0x380E0000, 0x380E4000, 0x380E8000, 0x380EC000, 0x380F0000, 0x380F4000, 0x380F8000,
0x380FC000,
01148      0x38100000, 0x38104000, 0x38108000, 0x3810C000, 0x38110000, 0x38114000, 0x38118000,
0x3811C000, 0x38120000, 0x38124000, 0x38128000, 0x3812C000, 0x38130000, 0x38134000, 0x38138000,
0x3813C000,
01149      0x38140000, 0x38144000, 0x38148000, 0x3814C000, 0x38150000, 0x38154000, 0x38158000,
0x3815C000, 0x38160000, 0x38164000, 0x38168000, 0x3816C000, 0x38170000, 0x38174000, 0x38178000,
0x3817C000,
01150      0x38180000, 0x38184000, 0x38188000, 0x3818C000, 0x38190000, 0x38194000, 0x38198000,
0x3819C000, 0x381A0000, 0x381A4000, 0x381A8000, 0x381AC000, 0x381B0000, 0x381B4000, 0x381B8000,
0x381BC000,
01151      0x381C0000, 0x381C4000, 0x381C8000, 0x381CC000, 0x381D0000, 0x381D4000, 0x381D8000,
0x381DC000, 0x381E0000, 0x381E4000, 0x381E8000, 0x381EC000, 0x381F0000, 0x381F4000, 0x381F8000,
0x381FC000,
01152      0x38200000, 0x38204000, 0x38208000, 0x3820C000, 0x38210000, 0x38214000, 0x38218000,
0x3821C000, 0x38220000, 0x38224000, 0x38228000, 0x3822C000, 0x38230000, 0x38234000, 0x38238000,
0x3823C000,
01153      0x38240000, 0x38244000, 0x38248000, 0x3824C000, 0x38250000, 0x38254000, 0x38258000,
0x3825C000, 0x38260000, 0x38264000, 0x38268000, 0x3826C000, 0x38270000, 0x38274000, 0x38278000,
```

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

```

0x03800000, 0x04000000, 0x04800000, 0x05000000, 0x05800000, 0x06000000, 0x06800000, 0x07000000,
0x07800000,
01242      0x08000000, 0x08800000, 0x09000000, 0x09800000, 0x0A000000, 0x0A800000, 0x0B000000,
0x0B800000, 0x0C000000, 0x0C800000, 0x0D000000, 0x0D800000, 0x0E000000, 0x0E800000, 0x0F000000,
0x47800000,
01243      0x80000000, 0x80800000, 0x81000000, 0x81800000, 0x82000000, 0x82800000, 0x83000000,
0x83800000, 0x84000000, 0x84800000, 0x85000000, 0x85800000, 0x86000000, 0x86800000, 0x87000000,
0x87800000,
01244      0x88000000, 0x88800000, 0x89000000, 0x89800000, 0x8A000000, 0x8A800000, 0x8B000000,
0x8B800000, 0x8C000000, 0x8C800000, 0x8D000000, 0x8D800000, 0x8E000000, 0x8E800000, 0x8F000000,
0xC7800000 };
01245      static const unsigned short offset_table[64] = {
01246          0, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024,
1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024,
01247          0, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024,
1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024,
01248      };
      bits<float>::type fbits = mantissa_table[offset_table[value>10]+(value&0x3FF)] +
      exponent_table[value>10];
01249      #endif
01250      float out;
01251      std::memcpy(&out, &fbits, sizeof(float));
01252      return out;
01253      #endif
01254      }
01255
01259      inline double half2float_impl(unsigned int value, double, true_type)
01260      {
01261      #if HALF_ENABLE_F16C_INTRINSICS
01262          return _mm_cvtsd_f64(_mm_cvtps_pd(_mm_cvtsi32_si128(value)));
01263      #else
01264          uint32 hi = static_cast<uint32>(value&0x8000) << 16;
01265          unsigned int abs = value & 0x7FFF;
01266          if(abs)
01267          {
01268              hi |= 0x3F000000 << static_cast<unsigned>(abs>=0x7C00);
01269              for(; abs<0x400; abs<<=1,hi-=0x100000) ;
01270              hi += static_cast<uint32>(abs) << 10;
01271          }
01272          bits<double>::type dbits = static_cast<bits<double>::type>(hi) << 32;
01273          double out;
01274          std::memcpy(&out, &dbits, sizeof(double));
01275          return out;
01276      #endif
01277      }
01278
01283      template<typename T> T half2float_impl(unsigned int value, T, ...)
01284      {
01285          T out;
01286          unsigned int abs = value & 0x7FFF;
01287          if(abs > 0x7C00)
01288              out = (std::numeric_limits<T>::has_signaling_NaN && !(abs&0x200)) ?
01289              std::numeric_limits<T>::signaling_NaN() :
01290              std::numeric_limits<T>::has_quiet_NaN ? std::numeric_limits<T>::quiet_NaN() : T();
01291          else if(abs == 0x7C00)
01292              out = std::numeric_limits<T>::has_infinity ? std::numeric_limits<T>::infinity() :
01293              std::numeric_limits<T>::max();
01294          else if(abs > 0x3FF)
01295              out = std::ldexp(static_cast<T>(abs&0x3FF|0x400), (abs>10)-25);
01296          else
01297              out = std::ldexp(static_cast<T>(abs), -24);
01298          return (value&0x8000) ? -out : out;
01299      }
01300
01303      template<typename T> T half2float(unsigned int value)
01304      {
01305          return half2float_impl(value, T(),
01306              bool_type<std::numeric_limits<T>::is_iec559&&sizeof(typename bits<T>::type)==sizeof(T)>());
01307      }
01308
01317      template<std::float_round_style R,bool E,bool I,typename T> T half2int(unsigned int value)
01318      {
01319          unsigned int abs = value & 0x7FFF;
01320          if(abs >= 0x7C00)
01321          {
01322              raise(FE_INVALID);
01323              return (value&0x8000) ? std::numeric_limits<T>::min() : std::numeric_limits<T>::max();
01324          }
01325          if(abs < 0x3800)
01326          {
01327              raise(FE_INEXACT, I);
01328              return (R==std::round_toward_infinity) ? T(~(value>15)&(abs!=0)) :
01329              (R==std::round_toward_neg_infinity) ? -T(value>0x8000) :
01330              T();
01331          }
01332          int exp = 25 - (abs>10);
01333          unsigned int m = (value&0x3FF) | 0x400;

```

```

01334         int32 i = static_cast<int32>((exp<=0) ? (m<<-exp) : (m+(
01335             (R==std::round_to_nearest) ? ((1<<(exp-1))- (~ (m>>exp)&E)) :
01336             (R==std::round_toward_infinity) ? ((1<<exp)-1)&((value>15)-1)) :
01337             (R==std::round_toward_neg_infinity) ? ((1<<exp)-1)&-(value>15)) : 0))>>exp));
01338         if(!std::numeric_limits<T>::is_signed && (value&0x8000) ||
01339             (std::numeric_limits<T>::digits<16 &&
01340                 ((value&0x8000) ? (-i<std::numeric_limits<T>::min()) :
01341                     (i>std::numeric_limits<T>::max()))))
01342             raise(FE_INVALID);
01343         else if(I && exp > 0 && (m&((1<<exp)-1)))
01344             raise(FE_INEXACT);
01345         return static_cast<T>((value&0x8000) ? -i : i);
01346     }
01347
01348     template<std::float_round_style R> uint32 mulhi(uint32 x, uint32 y)
01349     {
01350         uint32 xy = (x>>16) * (y&0xFFFF), yx = (x&0xFFFF) * (y>>16), c = (xy&0xFFFF) + (yx&0xFFFF) +
01351             ((x&0xFFFF)*(y&0xFFFF)>>16);
01352         return (x>>16)*(y>>16) + (xy>>16) + (yx>>16) + (c>>16) +
01353             ((R==std::round_to_nearest) ? ((c>>15)&1) : (R==std::round_toward_infinity) ?
01354             ((c&0xFFFF)!=0) : 0);
01355     }
01356
01357     inline uint32 multiply64(uint32 x, uint32 y)
01358     {
01359         #if HALF_ENABLE_CPP11_LONG_LONG
01360             return static_cast<uint32>((static_cast<unsigned long long>(x)*static_cast<unsigned long
01361             long>(y)+0x80000000)>>32);
01362         #else
01363             return mulhi<std::round_to_nearest>(x, y);
01364         #endif
01365     }
01366
01367     inline uint32 divide64(uint32 x, uint32 y, int &s)
01368     {
01369         #if HALF_ENABLE_CPP11_LONG_LONG
01370             unsigned long long xx = static_cast<unsigned long long>(x) << 32;
01371             return s = (xx%y!=0), static_cast<uint32>(xx/y);
01372         #else
01373             y >= 1;
01374             uint32 rem = x, div = 0;
01375             for(unsigned int i=0; i<32; ++i)
01376             {
01377                 div <= 1;
01378                 if(rem >= y)
01379                 {
01380                     rem -= y;
01381                     div |= 1;
01382                 }
01383                 rem <= 1;
01384             }
01385             return s = rem > 1, div;
01386         #endif
01387     }
01388
01389     template<bool Q,bool R> unsigned int mod(unsigned int x, unsigned int y, int *quo = NULL)
01390     {
01391         unsigned int q = 0;
01392         if(x > y)
01393         {
01394             int absx = x, absy = y, expx = 0, expy = 0;
01395             for(; absx<0x400; absx<=1,--expx) ;
01396             for(; absy<0x400; absy<=1,--expy) ;
01397             expx += absx > 10;
01398             expy += absy > 10;
01399             int mx = (absx&0x3FF) | 0x400, my = (absy&0x3FF) | 0x400;
01400             for(int d=expx-expy; d; --d)
01401             {
01402                 if(!Q && mx == my)
01403                     return 0;
01404                 if(mx >= my)
01405                 {
01406                     mx -= my;
01407                     q += Q;
01408                 }
01409                 mx <= 1;
01410                 q <= static_cast<int>(Q);
01411             }
01412             if(!Q && mx == my)
01413                 return 0;
01414             if(mx >= my)
01415             {
01416                 mx -= my;
01417                 ++q;
01418             }
01419             if(Q)

```

```

01440         {
01441             q &= (1<<(std::numeric_limits<int>::digits-1)) - 1;
01442             if(!mx)
01443                 return *quo = q, 0;
01444         }
01445         for(; mx<0x400; mx<=1,--expy) ;
01446         x = (expy>0) ? ((expy<10) ? (mx<0x3FF) : (mx<0x3FF)) : (mx<0x3FF);
01447     }
01448     if(R)
01449     {
01450         unsigned int a, b;
01451         if(y < 0x800)
01452         {
01453             a = (x<0x400) ? (x<1) : (x<0x400);
01454             b = y;
01455         }
01456         else
01457         {
01458             a = x;
01459             b = y - 0x400;
01460         }
01461         if(a > b || (a == b && (q&1)))
01462         {
01463             int exp = (y>10) + (y<=0x3FF), d = exp - (x>10) - (x<=0x3FF);
01464             int m = (((y<0x3FF) ? ((y>0x3FF)<10)<1) - (((x<0x3FF) ? ((x>0x3FF)<10)<1)<(1-d)));
01465             for(; m<0x800 && exp>1; m<=1,--exp) ;
01466             x = 0x8000 + ((exp-1)<10) + (m>1);
01467             q += Q;
01468         }
01469     }
01470     if(Q)
01471         *quo = q;
01472     return x;
01473 }
01474
01480 template<unsigned int F> uint32 sqrt(uint32 &r, int &exp)
01481 {
01482     int i = exp & 1;
01483     r <= i;
01484     exp = (exp-i) / 2;
01485     uint32 m = 0;
01486     for(uint32 bit=static_cast<uint32>(1)<<F; bit; bit<=2)
01487     {
01488         if(r < m+bit)
01489             m >= 1;
01490         else
01491         {
01492             r -= m + bit;
01493             m = (m>1) + bit;
01494         }
01495     }
01496     return m;
01497 }
01498
01500 inline uint32 exp2(uint32 m, unsigned int n = 32)
01501 {
01502     static const uint32 logs[] = {
01503         0x80000000, 0x4AE00D1D, 0x2934F098, 0x15C01A3A, 0x0B31FB7D, 0x05AEB4DD, 0x02DCF2D1,
01504         0x016FE50B,
01505         0x00B84E23, 0x005C3E10, 0x002E24CA, 0x001713D6, 0x000B8A47, 0x0005C53B, 0x0002E2A3,
01506         0x00017153,
01507         0x0000B8AA, 0x00005C55, 0x00002E2B, 0x00001715, 0x00000B8B, 0x000005C5, 0x000002E3,
01508         0x00000171,
01509         0x000000B9, 0x0000005C, 0x0000002E, 0x00000017, 0x0000000C, 0x00000006, 0x00000003,
01510         0x00000001 };
01511     if(!m)
01512         return 0x80000000;
01513     uint32 mx = 0x80000000, my = 0;
01514     for(unsigned int i=1; i<n; ++i)
01515     {
01516         uint32 mz = my + logs[i];
01517         if(mz <= m)
01518         {
01519             my = mz;
01520             mx += mx >> i;
01521         }
01522     }
01523     return mx;
01524 }
01525
01531 inline uint32 log2(uint32 m, unsigned int n = 32)
01532 {
01533     static const uint32 logs[] = {
01534         0x80000000, 0x4AE00D1D, 0x2934F098, 0x15C01A3A, 0x0B31FB7D, 0x05AEB4DD, 0x02DCF2D1,
01535         0x016FE50B,
01536         0x00B84E23, 0x005C3E10, 0x002E24CA, 0x001713D6, 0x000B8A47, 0x0005C53B, 0x0002E2A3,
01537         0x00017153,
01538         0x0000B8AA, 0x00005C55, 0x00002E2B, 0x00001715, 0x00000B8B, 0x000005C5, 0x000002E3,
01539         0x00000171,
01540         0x000000B9, 0x0000005C, 0x0000002E, 0x00000017, 0x0000000C, 0x00000006, 0x00000003,
01541         0x00000001 };
01542     if(!m)
01543         return 0x80000000;
01544     uint32 mx = 0x80000000, my = 0;
01545     for(unsigned int i=1; i<n; ++i)
01546     {
01547         uint32 mz = my + logs[i];
01548         if(mz <= m)
01549         {
01550             my = mz;
01551             mx += mx >> i;
01552         }
01553     }
01554     return mx;
01555 }

```

```

01536         0x0000B8AA, 0x00005C55, 0x00002E2B, 0x00001715, 0x00000B8B, 0x000005C5, 0x000002E3,
01537         0x00000171,
01538         0x000000B9, 0x0000005C, 0x0000002E, 0x00000017, 0x0000000C, 0x00000006, 0x00000003,
01539         0x00000001 };
01540         if(m == 0x40000000)
01541             return 0;
01542         uint32 mx = 0x40000000, my = 0;
01543         for(unsigned int i=1; i<n; ++i)
01544         {
01545             uint32 mz = mx + (mx>>i);
01546             if(mz <= m)
01547             {
01548                 mx = mz;
01549                 my += logs[i];
01550             }
01551         }
01552         return my;
01553
01554         inline std::pair<uint32,uint32> sincos(uint32 mz, unsigned int n = 31)
01555         {
01556             static const uint32 angles[] = {
01557                 0x3243F6A9, 0x1DAC6705, 0x0FADBAFD, 0x07F56EA7, 0x03FEAB77, 0x01FFD55C, 0x00FFFAAB,
01558                 0x007FFF55,
01559                 0x003FFFEb, 0x001FFFFD, 0x00100000, 0x00080000, 0x00040000, 0x00020000, 0x00010000,
01560                 0x00008000,
01561                 0x00004000, 0x00002000, 0x00001000, 0x00000800, 0x00000400, 0x00000200, 0x00000100,
01562                 0x00000080,
01563                 0x00000040, 0x00000020, 0x00000010, 0x00000008, 0x00000004, 0x00000002, 0x00000001 };
01564             uint32 mx = 0x2DD3B6A, my = 0;
01565             for(unsigned int i=0; i<n; ++i)
01566             {
01567                 uint32 sign = sign_mask(mz);
01568                 uint32 tx = mx - (arithmetic_shift(my, i)^sign) + sign;
01569                 uint32 ty = my + (arithmetic_shift(mx, i)^sign) - sign;
01570                 mx = tx; my = ty; mz -= (angles[i]^sign) - sign;
01571             }
01572             return std::make_pair(my, mx);
01573         }
01574
01575         inline uint32 atan2(uint32 my, uint32 mx, unsigned int n = 31)
01576         {
01577             static const uint32 angles[] = {
01578                 0x3243F6A9, 0x1DAC6705, 0x0FADBAFD, 0x07F56EA7, 0x03FEAB77, 0x01FFD55C, 0x00FFFAAB,
01579                 0x007FFF55,
01580                 0x003FFFEb, 0x001FFFFD, 0x00100000, 0x00080000, 0x00040000, 0x00020000, 0x00010000,
01581                 0x00008000,
01582                 0x00004000, 0x00002000, 0x00001000, 0x00000800, 0x00000400, 0x00000200, 0x00000100,
01583                 0x00000080,
01584                 0x00000040, 0x00000020, 0x00000010, 0x00000008, 0x00000004, 0x00000002, 0x00000001 };
01585             uint32 mz = 0;
01586             for(unsigned int i=0; i<n; ++i)
01587             {
01588                 uint32 sign = sign_mask(my);
01589                 uint32 tx = mx + (arithmetic_shift(my, i)^sign) - sign;
01590                 uint32 ty = my - (arithmetic_shift(mx, i)^sign) + sign;
01591                 mx = tx; my = ty; mz += (angles[i]^sign) - sign;
01592             }
01593             return mz;
01594         }
01595
01596         inline uint32 angle_arg(unsigned int abs, int &k)
01597         {
01598             uint32 m = (abs&0x3FF) | ((abs>0x3FF)<<10);
01599             int exp = (abs>>10) + (abs<=0x3FF) - 15;
01600             if(abs < 0x3A48)
01601                 return k = 0, m << (exp+20);
01602             #if HALF_ENABLE_CPP11_LONG_LONG
01603             unsigned long long y = m * 0xA2F9836E4E442, mask = (1ULL<<(62-exp)) - 1, yi = (y+(mask>>1))
01604             & ~mask, f = y - yi;
01605             uint32 sign = -static_cast<uint32>(f>>63);
01606             k = static_cast<int>(yi>>(62-exp));
01607             return (multiply64(static_cast<uint32>((sign ? -f : f)>>(31-exp)), 0xC90FDAA2)^sign) -
01608             sign;
01609             #else
01610             uint32 yh = m*0xA2F98 + mulhi<std::round_toward_zero>(m, 0x36E4E442), yl = (m*0x36E4E442)
01611             & 0xFFFFFFFF;
01612             uint32 mask = (static_cast<uint32>(1)<<(30-exp)) - 1, yi = (yh+(mask>>1)) & ~mask, sign =
01613             -static_cast<uint32>(yi>yh);
01614             k = static_cast<int>(yi>>(30-exp));
01615             uint32 fh = (yh^sign) + (yi^~sign) - ~sign, fl = (yl^sign) - sign;
01616             return (multiply64((exp>-1) ? (((fh<<(1+exp))&0xFFFFFFFF)|((fl&0xFFFFFFFF)>>(31-exp))) : fh,
01617             0xC90FDAA2)^sign) - sign;
01618             #endif
01619         }
01620
01621         inline std::pair<uint32,uint32> atan2_args(unsigned int abs)

```



```

01628     {
01629         int exp = -15;
01630         for(; abs<0x400; abs<=1,--exp) ;
01631         exp += abs >> 10;
01632         uint32 my = ((abs&0x3FF)|0x400) << 5, r = my * my;
01633         int rexp = 2 * exp;
01634         r = 0x40000000 - ((rexp>-31) ? ((r>-rexp)|((r&((static_cast<uint32>(1)<<-rexp)-1))!=0)) :
1);
01635         for(rexp=0; r<0x40000000; r<=1,--rexp) ;
01636         uint32 mx = sqrt<30>(r, rexp);
01637         int d = exp - rexp;
01638         if(d < 0)
01639             return std::make_pair((d<-14) ? ((my)>(-d-14))+((my)>(-d-15))&1) : (my<(14+d)),
(mx<(14)+(r<(13)/mx));
01640         if(d > 0)
01641             return std::make_pair(my<14, (d>14) ? ((mx)>(d-14))+((mx)>(d-15))&1) : ((d==14) ? mx :
((mx<(14-d))+((r<(13-d))/mx)));
01642         return std::make_pair(my<13, (mx<13)+(r<12)/mx);
01643     }
01644
01650     inline std::pair<uint32,uint32> hyperbolic_args(unsigned int abs, int &exp, unsigned int n =
32)
01651     {
01652         uint32 mx = detail::multiply64(static_cast<uint32>((abs&0x3FF)+((abs>0x3FF)<<10))<<21,
0xB8AA3B29), my;
01653         int e = (abs>10) + (abs<=0x3FF);
01654         if(e < 14)
01655         {
01656             exp = 0;
01657             mx >>= 14 - e;
01658         }
01659         else
01660         {
01661             exp = mx >> (45-e);
01662             mx = (mx<(e-14)) & 0x7FFFFFFF;
01663         }
01664         mx = exp2(mx, n);
01665         int d = exp << 1, s;
01666         if(mx > 0x80000000)
01667         {
01668             my = divide64(0x80000000, mx, s);
01669             my |= s;
01670             ++d;
01671         }
01672         else
01673             my = mx;
01674         return std::make_pair(mx, (d<31) ? ((my)>d)|((my&((static_cast<uint32>(1)<<d)-1))!=0) : 1);
01675     }
01676
01688     template<std::float_round_style R> unsigned int exp2_post(uint32 m, int exp, bool esign,
unsigned int sign = 0, unsigned int n = 32)
01689     {
01690         if(esign)
01691         {
01692             exp = -exp - (m!=0);
01693             if(exp < -25)
01694                 return underflow<R>(sign);
01695             else if(exp == -25)
01696                 return rounded<R,false>(sign, 1, m!=0);
01697         }
01698         else if(exp > 15)
01699             return overflow<R>(sign);
01700         if(!m)
01701             return sign | (((exp+=15)>0) ? (exp<10) : check_underflow(0x200>-exp));
01702         m = exp2(m, n);
01703         int s = 0;
01704         if(esign)
01705             m = divide64(0x80000000, m, s);
01706         return fixed2half<R,31,false,false,true>(m, exp+14, sign, s);
01707     }
01708
01720     template<std::float_round_style R,uint32 L> unsigned int log2_post(uint32 m, int ilog, int
exp, unsigned int sign = 0)
01721     {
01722         uint32 msign = sign_mask(ilog);
01723         m = (((static_cast<uint32>(ilog)<<27)+(m>4))<^msign) - msign;
01724         if(!m)
01725             return 0;
01726         for(; m<0x80000000; m<=1,--exp) ;
01727         int i = m >= L, s;
01728         exp += i;
01729         m >>= 1 + i;
01730         sign ^= msign & 0x8000;
01731         if(exp < -11)
01732             return underflow<R>(sign);
01733         m = divide64(m, L, s);
01734         return fixed2half<R,30,false,false,true>(m, exp, sign, 1);

```

```

01735     }
01736
01745     template<std::float_round_style R> unsigned int hypot_post(uint32 r, int exp)
01746     {
01747         int i = r >> 31;
01748         if((exp+i) > 46)
01749             return overflow<R>();
01750         if(exp < -34)
01751             return underflow<R>();
01752         r = (r>i) | (r&i);
01753         uint32 m = sqrt<30>(r, exp+=15);
01754         return fixed2half<R,15,false,false,false>(m, exp-1, 0, r!=0);
01755     }
01756
01767     template<std::float_round_style R> unsigned int tangent_post(uint32 my, uint32 mx, int exp,
unsigned int sign = 0)
01768     {
01769         int i = my >= mx, s;
01770         exp += i;
01771         if(exp > 29)
01772             return overflow<R>(sign);
01773         if(exp < -11)
01774             return underflow<R>(sign);
01775         uint32 m = divide64(my>>(i+1), mx, s);
01776         return fixed2half<R,30,false,false,true>(m, exp, sign, s);
01777     }
01778
01788     template<std::float_round_style R,bool S> unsigned int area(unsigned int arg)
01789     {
01790         int abs = arg & 0x7FFF, expx = (abs>>10) + (abs<=0x3FFF) - 15, expy = -15, ilog, i;
01791         uint32 mx = static_cast<uint32>((abs&0x3FFF)|((abs>0x3FFF)<<10)) << 20, my, r;
01792         for(; abs<0x400; abs<<=1,--expy) ;
01793         expy += abs >> 10;
01794         r = ((abs&0x3FFF)|0x400) << 5;
01795         r *= r;
01796         i = r >> 31;
01797         expy = 2*expy + i;
01798         r >= i;
01799         if(S)
01800         {
01801             if(expy < 0)
01802             {
01803                 r = 0x40000000 + ((expy>-30) ?
((r>=expy)|((r&((static_cast<uint32>(1)<<-expy)-1))!=0)) : 1);
01804                 expy = 0;
01805             }
01806             else
01807             {
01808                 r += 0x40000000 >> expy;
01809                 i = r >> 31;
01810                 r = (r>i) | (r&i);
01811                 expy += i;
01812             }
01813         }
01814         else
01815         {
01816             r -= 0x40000000 >> expy;
01817             for(; r<0x40000000; r<<=1,--expy) ;
01818         }
01819         my = sqrt<30>(r, expy);
01820         my = (my<<15) + (r<<14)/my;
01821         if(S)
01822         {
01823             mx >= expy - expx;
01824             ilog = expy;
01825         }
01826         else
01827         {
01828             my >= expx - expy;
01829             ilog = expx;
01830         }
01831         my += mx;
01832         i = my >> 31;
01833         static const int G = S && (R==std::round_to_nearest);
01834         return log2_post<R,0xB8AA3B2A>(log2(my>>i, 26+S+G)+(G<3), ilog+i, 17,
arg&(static_cast<unsigned>(S)<<15));
01835     }
01836
01837     struct f31
01838     {
01839         HALF_CONSTEXPR f31(uint32 mant, int e) : m(mant), exp(e) {}
01840
01841         f31(unsigned int abs) : exp(-15)
01842         {
01843             for(; abs<0x400; abs<<=1,--exp) ;
01844             m = static_cast<uint32>((abs&0x3FFF)|0x400) << 21;
01845             exp += (abs>>10);

```

```

01852     }
01853
01858     friend f31 operator+(f31 a, f31 b)
01859     {
01860         if(b.exp > a.exp)
01861             std::swap(a, b);
01862         int d = a.exp - b.exp;
01863         uint32 m = a.m + ((d<32) ? (b.m>>d) : 0);
01864         int i = (m&0xFFFFFFFF) < a.m;
01865         return f31(((m+i)>>i)|0x80000000, a.exp+i);
01866     }
01867
01872     friend f31 operator-(f31 a, f31 b)
01873     {
01874         int d = a.exp - b.exp, exp = a.exp;
01875         uint32 m = a.m - ((d<32) ? (b.m>>d) : 0);
01876         if(!m)
01877             return f31(0, -32);
01878         for(; m<0x80000000; m<=1,--exp) ;
01879         return f31(m, exp);
01880     }
01881
01886     friend f31 operator*(f31 a, f31 b)
01887     {
01888         uint32 m = multiply64(a.m, b.m);
01889         int i = m >> 31;
01890         return f31(m<<(1-i), a.exp + b.exp + i);
01891     }
01892
01897     friend f31 operator/(f31 a, f31 b)
01898     {
01899         int i = a.m >= b.m, s;
01900         uint32 m = divide64((a.m+i)>>i, b.m, s);
01901         return f31(m, a.exp - b.exp + i - 1);
01902     }
01903
01904     uint32 m;
01905     int exp;
01906 };
01907
01918     template<std::float_round_style R,bool C> unsigned int erf(unsigned int arg)
01919     {
01920         unsigned int abs = arg & 0xFFFF, sign = arg & 0x8000;
01921         f31 x(abs), x2 = x * x * f31(0xB8AA3B29, 0), t = f31(0x80000000, 0) / (f31(0x80000000,
01922 0)+f31(0xA7BA054A, -2)*x), t2 = t * t;
01923         f31 e = ((f31(0x87DC2213, 0)*t2+f31(0xB5F0E2AE, 0))*t2+f31(0x82790637,
01924 -2)-(f31(0xBA00E2B8, 0)*t2+f31(0x91A98E62, -2))*t) * t /
01925 ((x2.exp<0) ? f31(exp2((x2.exp>-32) ? (x2.m)>>-x2.exp) : 0, 30), 0) :
01926 f31(exp2((x2.m<>x2.exp)&0x7FFFFFFF, 22), x2.m)>>(31-x2.exp));
01927         return (!C || sign) ? fixed2half<R,31,false,true,true>(0x80000000-(e.m)>>(C-e.exp)), 14+C,
01928 sign&(C-1U)) :
01929 (e.exp<-25) ? underflow<R>() : fixed2half<R,30,false,false,true>(e.m)>>1, e.exp+14,
01930 0, e.m&1);
01931     }
01932
01937     template<std::float_round_style R,bool L> unsigned int gamma(unsigned int arg)
01938     {
01939         /* static const double p[] = { 2.50662827563479526904, 225.525584619175212544,
01940 -268.295973841304927459, 80.9030806934622512966, -5.00757863970517583837, 0.0114684895434781459556 };
01941         double t = arg + 4.65, s = p[0];
01942         for(unsigned int i=0; i<5; ++i)
01943             s += p[i+1] / (arg+i);
01944         return std::log(s) + (arg-0.5)*std::log(t) - t;
01945         static const f31 pi(0xC90FDA2, 1), lbe(0xB8AA3B29, 0);
01946         unsigned int abs = arg & 0xFFFF, sign = arg & 0x8000;
01947         bool bsign = sign != 0;
01948         f31 z(abs), x = sign ? (z+f31(0x80000000, 0)) : z, t = x + f31(0x94CCCCD, 2), s =
01949 f31(0xA06C9901, 1) + f31(0xBBE654E2, -7)/(x+f31(0x80000000, 2)) + f31(0xA1CE6098,
01950 6)/(x+f31(0x80000000, 1))
01951 + f31(0xE1868CB7, 7)/x - f31(0x8625E279, 8)/(x+f31(0x80000000, 0)) - f31(0xA03E158F,
01952 2)/(x+f31(0xC0000000, 1));
01953         int i = (s.exp>=2) + (s.exp>=4) + (s.exp>=8) + (s.exp>=16);
01954         s = f31((static_cast<uint32>(s.exp)<<(31-i))+(log2(s.m)>>1, 28)>>i), i) / lbe;
01955         if(x.exp != -1 || x.m != 0x80000000)
01956         {
01957             i = (t.exp>=2) + (t.exp>=4) + (t.exp>=8);
01958             f31 l = f31((static_cast<uint32>(t.exp)<<(31-i))+(log2(t.m)>>1, 30)>>i), i) / lbe;
01959             s = (x.exp<-1) ? (s-(f31(0x80000000, -1)-x)*l) : (s+(x-f31(0x80000000, -1))*l);
01960         }
01961         s = x.exp ? (s-t) : (t-s);
01962         if(bsign)
01963         {
01964             if(z.exp >= 0)
01965             {
01966                 sign &= (L|((z.m)>>(31-z.exp))&1) - 1;
01967                 for(z=f31((z.m<<(1+z.exp))&0xFFFFFFFF, -1); z.m<0x80000000; z.m<=1,--z.exp) ;
01968             }
01969         }

```

```

01966         if(z.exp == -1)
01967             z = f31(0x80000000, 0) - z;
01968         if(z.exp < -1)
01969         {
01970             z = z * pi;
01971             z.m = sincos(z.m»(1-z.exp), 30).first;
01972             for(z.exp=1; z.m<0x80000000; z.m«=1,--z.exp) ;
01973         }
01974         else
01975             z = f31(0x80000000, 0);
01976     }
01977     if(L)
01978     {
01979         if(bsign)
01980         {
01981             f31 l(0x92868247, 0);
01982             if(z.exp < 0)
01983             {
01984                 uint32 m = log2((z.m+1)»1, 27);
01985                 z = f31(-(static_cast<uint32>(z.exp«26)+(m»5)), 5);
01986                 for(; z.m<0x80000000; z.m«=1,--z.exp) ;
01987                 l = l + z / lbe;
01988             }
01989             sign = static_cast<unsigned>(x.exp&&(l.exp<s.exp||(l.exp==s.exp&&l.m<s.m))) « 15;
01990             s = sign ? (s-1) : x.exp ? (l-s) : (l+s);
01991         }
01992         else
01993         {
01994             sign = static_cast<unsigned>(x.exp==0) « 15;
01995             if(s.exp < -24)
01996                 return underflow<R>(sign);
01997             if(s.exp > 15)
01998                 return overflow<R>(sign);
01999         }
02000     }
02001     else
02002     {
02003         s = s * lbe;
02004         uint32 m;
02005         if(s.exp < 0)
02006         {
02007             m = s.m » -s.exp;
02008             s.exp = 0;
02009         }
02010         else
02011         {
02012             m = (s.m«s.exp) & 0x7FFFFFFF;
02013             s.exp = (s.m»(31-s.exp));
02014         }
02015         s.m = exp2(m, 27);
02016         if(!x.exp)
02017             s = f31(0x80000000, 0) / s;
02018         if(bsign)
02019         {
02020             if(z.exp < 0)
02021                 s = s * z;
02022             s = pi / s;
02023             if(s.exp < -24)
02024                 return underflow<R>(sign);
02025         }
02026         else if(z.exp > 0 && !(z.m&((1«(31-z.exp))-1)))
02027             return ((s.exp+14)«10) + (s.m»21);
02028         if(s.exp > 15)
02029             return overflow<R>(sign);
02030     }
02031     return fixed2half<R,31,false,false,true>(s.m, s.exp+14, sign);
02032 }
02033
02034 template<typename,typename,std::float_round_style> struct half_caster;
02035
02036 }
02037
02038 class half
02039 {
02040 public:
02041     HALF_CONSTEXPR half() HALF_NOEXCEPT : data_() {}
02042
02043     explicit half(float rhs) :
02044         data_(static_cast<detail::uint16>(detail::float2half<round_style>(rhs))) {}
02045
02046     operator float() const { return detail::half2float<float>(data_); }
02047
02048     half& operator=(float rhs) { data_ =
02049         static_cast<detail::uint16>(detail::float2half<round_style>(rhs)); return *this; }
02050
02051     half& operator+=(half rhs) { return *this = *this + rhs; }

```

```

02091
02097     half& operator--(half rhs) { return *this = *this - rhs; }
02098
02104     half& operator*=(half rhs) { return *this = *this * rhs; }
02105
02111     half& operator/=(half rhs) { return *this = *this / rhs; }
02112
02117     half& operator+=(float rhs) { return *this = *this + rhs; }
02118
02123     half& operator--(float rhs) { return *this = *this - rhs; }
02124
02129     half& operator*=(float rhs) { return *this = *this * rhs; }
02130
02135     half& operator/=(float rhs) { return *this = *this / rhs; }
02136
02140
02144     half& operator++() { return *this = *this + half(detail::binary, 0x3C00); }
02145
02149     half& operator--() { return *this = *this + half(detail::binary, 0xBC00); }
02150
02154     half operator++(int) { half out(*this); ++*this; return out; }
02155
02159     half operator--(int) { half out(*this); --*this; return out; }
02161
02162     private:
02164         static const std::float_round_style round_style = (std::float_round_style)(HALF_ROUND_STYLE);
02165
02168         HALF_CONSTEXPR half(detail::binary_t, unsigned int bits) HALF_NOEXCEPT :
02169             data_(static_cast<detail::uint16>(bits)) {}
02170
02171         detail::uint16 data_;
02172
02173         #ifndef HALF_DOXYGEN_ONLY
02174             friend HALF_CONSTEXPR_NOERR bool operator==(half, half);
02175             friend HALF_CONSTEXPR_NOERR bool operator!=(half, half);
02176             friend HALF_CONSTEXPR_NOERR bool operator<(half, half);
02177             friend HALF_CONSTEXPR_NOERR bool operator>(half, half);
02178             friend HALF_CONSTEXPR_NOERR bool operator<=(half, half);
02179             friend HALF_CONSTEXPR_NOERR bool operator>=(half, half);
02180             friend HALF_CONSTEXPR half operator-(half);
02181             friend half operator+(half, half);
02182             friend half operator-(half, half);
02183             friend half operator*(half, half);
02184             friend half operator/(half, half);
02185             template<typename charT, typename traits> friend std::basic_ostream<charT, traits>&
02186                 operator<<(std::basic_ostream<charT, traits>&, half);
02187             template<typename charT, typename traits> friend std::basic_istream<charT, traits>&
02188                 operator>>(std::basic_istream<charT, traits>&, half);
02189             friend HALF_CONSTEXPR half fabs(half);
02190             friend half fmod(half, half);
02191             friend half remainder(half, half);
02192             friend half remquo(half, half, int*);
02193             friend half fma(half, half, half);
02194             friend HALF_CONSTEXPR_NOERR half fmax(half, half);
02195             friend HALF_CONSTEXPR_NOERR half fmin(half, half);
02196             friend half fdim(half, half);
02197             friend half nanh(const char*);
02198             friend half exp(half);
02199             friend half exp2(half);
02200             friend half expm1(half);
02201             friend half log(half);
02202             friend half log10(half);
02203             friend half log2(half);
02204             friend half log1p(half);
02205             friend half sqrt(half);
02206             friend half rsqrt(half);
02207             friend half cbrt(half);
02208             friend half hypot(half, half);
02209             friend half hypot(half, half, half);
02210             friend half pow(half, half);
02211             friend void sincos(half, half*, half*);
02212             friend half sin(half);
02213             friend half cos(half);
02214             friend half tan(half);
02215             friend half asin(half);
02216             friend half acos(half);
02217             friend half atan(half);
02218             friend half atan2(half, half);
02219             friend half sinh(half);
02220             friend half cosh(half);
02221             friend half tanh(half);
02222             friend half asinh(half);
02223             friend half acosh(half);
02224             friend half atanh(half);
02225             friend half erf(half);
02226             friend half erfc(half);
02227             friend half lgamma(half);

```

```

02226     friend half tgamma(half);
02227     friend half ceil(half);
02228     friend half floor(half);
02229     friend half trunc(half);
02230     friend half round(half);
02231     friend long lround(half);
02232     friend half rint(half);
02233     friend long lrint(half);
02234     friend half nearbyint(half);
02235 #ifdef HALF_ENABLE_CPP11_LONG_LONG
02236     friend long long llround(half);
02237     friend long long llrint(half);
02238 #endif
02239     friend half frexp(half, int*);
02240     friend half scalbn(half, long);
02241     friend half modf(half, half*);
02242     friend int ilogb(half);
02243     friend half logb(half);
02244     friend half nextafter(half, half);
02245     friend half nexttoward(half, long double);
02246     friend HALF_CONSTEXPR half copysign(half, half);
02247     friend HALF_CONSTEXPR int fpclassify(half);
02248     friend HALF_CONSTEXPR bool isfinite(half);
02249     friend HALF_CONSTEXPR bool isinf(half);
02250     friend HALF_CONSTEXPR bool isnan(half);
02251     friend HALF_CONSTEXPR bool isnormal(half);
02252     friend HALF_CONSTEXPR bool signbit(half);
02253     friend HALF_CONSTEXPR bool isgreater(half, half);
02254     friend HALF_CONSTEXPR bool isgreaterequal(half, half);
02255     friend HALF_CONSTEXPR bool isless(half, half);
02256     friend HALF_CONSTEXPR bool islessequal(half, half);
02257     friend HALF_CONSTEXPR bool islessgreater(half, half);
02258     template<typename, typename, std::float_round_style> friend struct detail::half_caster;
02259     friend class std::numeric_limits<half>;
02260 #if HALF_ENABLE_CPP11_HASH
02261     friend struct std::hash<half>;
02262 #endif
02263 #if HALF_ENABLE_CPP11_USER_LITERALS
02264     friend half literal::operator "" _h(long double);
02265 #endif
02266 #endif
02267 };
02268
02269 #if HALF_ENABLE_CPP11_USER_LITERALS
02270 namespace literal
02271 {
02272     inline half operator "" _h(long double value) { return half(detail::binary,
02273 detail::float2half<half::round_style>(value)); }
02274 }
02275 #endif
02276
02277 namespace detail
02278 {
02279     template<typename T, typename U, std::float_round_style
02280 R=(std::float_round_style) (HALF_ROUND_STYLE)> struct half_caster {};
02281     {
02282         #if HALF_ENABLE_CPP11_STATIC_ASSERT && HALF_ENABLE_CPP11_TYPE_TRAITS
02283             static_assert(std::is_arithmetic<U>::value, "half_cast from non-arithmetic type
02284 unsupported");
02285         #endif
02286
02287         static half cast(U arg) { return cast_impl(arg, is_float<U>()); };
02288
02289     private:
02290         static half cast_impl(U arg, true_type) { return half(binary, float2half<R>(arg)); }
02291         static half cast_impl(U arg, false_type) { return half(binary, int2half<R>(arg)); }
02292     };
02293     template<typename T, std::float_round_style R> struct half_caster<T, half, R>
02294     {
02295         #if HALF_ENABLE_CPP11_STATIC_ASSERT && HALF_ENABLE_CPP11_TYPE_TRAITS
02296             static_assert(std::is_arithmetic<T>::value, "half_cast to non-arithmetic type
02297 unsupported");
02298         #endif
02299
02300         static T cast(half arg) { return cast_impl(arg, is_float<T>()); }
02301
02302     private:
02303         static T cast_impl(half arg, true_type) { return half2float<T>(arg.data_); }
02304         static T cast_impl(half arg, false_type) { return half2int<R, true, true, T>(arg.data_); }
02305     };
02306     template<std::float_round_style R> struct half_caster<half, half, R>
02307     {
02308         static half cast(half arg) { return arg; }
02309     };
02310 }
02311
02312 #endif
02313 }
02314 }
02315 }
02316 }
02317 }
02318 }
02319 }
02320 }
02321 }

```

```

02322
02324 namespace std
02325 {
02328     template<> class numeric_limits<half_float::half>
02329     {
02330     public:
02332         static HALF_CONSTEXPR_CONST bool is_specialized = true;
02333
02335         static HALF_CONSTEXPR_CONST bool is_signed = true;
02336
02338         static HALF_CONSTEXPR_CONST bool is_integer = false;
02339
02341         static HALF_CONSTEXPR_CONST bool is_exact = false;
02342
02344         static HALF_CONSTEXPR_CONST bool is_modulo = false;
02345
02347         static HALF_CONSTEXPR_CONST bool is_bounded = true;
02348
02350         static HALF_CONSTEXPR_CONST bool is_iec559 = true;
02351
02353         static HALF_CONSTEXPR_CONST bool has_infinity = true;
02354
02356         static HALF_CONSTEXPR_CONST bool has_quiet_NaN = true;
02357
02359         static HALF_CONSTEXPR_CONST bool has_signaling_NaN = true;
02360
02362         static HALF_CONSTEXPR_CONST float_denorm_style has_denorm = denorm_present;
02363
02365         static HALF_CONSTEXPR_CONST bool has_denorm_loss = false;
02366
02367         #if HALF_ERRHANDLING_THROWS
02368             static HALF_CONSTEXPR_CONST bool traps = true;
02369         #else
02371             static HALF_CONSTEXPR_CONST bool traps = false;
02372         #endif
02373
02375         static HALF_CONSTEXPR_CONST bool tinyness_before = false;
02376
02378         static HALF_CONSTEXPR_CONST float_round_style round_style = half_float::half::round_style;
02379
02381         static HALF_CONSTEXPR_CONST int digits = 11;
02382
02384         static HALF_CONSTEXPR_CONST int digits10 = 3;
02385
02387         static HALF_CONSTEXPR_CONST int max_digits10 = 5;
02388
02390         static HALF_CONSTEXPR_CONST int radix = 2;
02391
02393         static HALF_CONSTEXPR_CONST int min_exponent = -13;
02394
02396         static HALF_CONSTEXPR_CONST int min_exponent10 = -4;
02397
02399         static HALF_CONSTEXPR_CONST int max_exponent = 16;
02400
02402         static HALF_CONSTEXPR_CONST int max_exponent10 = 4;
02403
02405         static HALF_CONSTEXPR half_float::half min() HALF_NOTHROW { return
half_float::half(half_float::detail::binary, 0x0400); }
02406
02408         static HALF_CONSTEXPR half_float::half lowest() HALF_NOTHROW { return
half_float::half(half_float::detail::binary, 0xFBFF); }
02409
02411         static HALF_CONSTEXPR half_float::half max() HALF_NOTHROW { return
half_float::half(half_float::detail::binary, 0x7BFF); }
02412
02414         static HALF_CONSTEXPR half_float::half epsilon() HALF_NOTHROW { return
half_float::half(half_float::detail::binary, 0x1400); }
02415
02417         static HALF_CONSTEXPR half_float::half round_error() HALF_NOTHROW
02418         { return half_float::half(half_float::detail::binary, (round_style==std::round_to_nearest)
? 0x3800 : 0x3C00); }
02419
02421         static HALF_CONSTEXPR half_float::half infinity() HALF_NOTHROW { return
half_float::half(half_float::detail::binary, 0x7C00); }
02422
02424         static HALF_CONSTEXPR half_float::half quiet_NaN() HALF_NOTHROW { return
half_float::half(half_float::detail::binary, 0x7FFF); }
02425
02427         static HALF_CONSTEXPR half_float::half signaling_NaN() HALF_NOTHROW { return
half_float::half(half_float::detail::binary, 0x7DFF); }
02428
02430         static HALF_CONSTEXPR half_float::half denorm_min() HALF_NOTHROW { return
half_float::half(half_float::detail::binary, 0x0001); }
02431     };
02432
02433     #if HALF_ENABLE_CPP11_HASH
02438         template<> struct hash<half_float::half>

```

```

02439     {
02441         typedef half_float::half argument_type;
02442
02444         typedef size_t result_type;
02445
02449         result_type operator() (argument_type arg) const { return
hash<half_float::detail::uint16>() (arg.data_&-static_cast<unsigned>(arg.data_!=0x8000)); }
02450     };
02451 #endif
02452 }
02453
02454 namespace half_float
02455 {
02459     inline HALF_CONSTEXPR_NOERR bool operator==(half x, half y)
02467     {
02468         return !detail::compsignal(x.data_, y.data_) && (x.data_==y.data_ ||
!((x.data_|y.data_)&0x7FFF));
02469     }
02470
02477     inline HALF_CONSTEXPR_NOERR bool operator!=(half x, half y)
02478     {
02479         return detail::compsignal(x.data_, y.data_) || (x.data_!=y.data_ &&
((x.data_|y.data_)&0x7FFF));
02480     }
02481
02488     inline HALF_CONSTEXPR_NOERR bool operator<(half x, half y)
02489     {
02490         return !detail::compsignal(x.data_, y.data_) &&
((x.data_^(0x8000|(0x8000-(x.data_>15))))+(x.data_>15)) <
((y.data_^(0x8000|(0x8000-(y.data_>15))))+(y.data_>15));
02492     }
02493
02500     inline HALF_CONSTEXPR_NOERR bool operator>(half x, half y)
02501     {
02502         return !detail::compsignal(x.data_, y.data_) &&
((x.data_^(0x8000|(0x8000-(x.data_>15))))+(x.data_>15)) >
((y.data_^(0x8000|(0x8000-(y.data_>15))))+(y.data_>15));
02504     }
02505
02512     inline HALF_CONSTEXPR_NOERR bool operator<=(half x, half y)
02513     {
02514         return !detail::compsignal(x.data_, y.data_) &&
((x.data_^(0x8000|(0x8000-(x.data_>15))))+(x.data_>15)) <=
((y.data_^(0x8000|(0x8000-(y.data_>15))))+(y.data_>15));
02516     }
02517
02524     inline HALF_CONSTEXPR_NOERR bool operator>=(half x, half y)
02525     {
02526         return !detail::compsignal(x.data_, y.data_) &&
((x.data_^(0x8000|(0x8000-(x.data_>15))))+(x.data_>15)) >=
((y.data_^(0x8000|(0x8000-(y.data_>15))))+(y.data_>15));
02528     }
02529
02534     inline HALF_CONSTEXPR half operator+(half arg) { return arg; }
02539
02543     inline HALF_CONSTEXPR half operator-(half arg) { return half(detail::binary, arg.data_^0x8000); }
02544
02552     inline half operator+(half x, half y)
02553     {
02554         #ifdef HALF_ARITHMETIC_TYPE
02555             return half(detail::binary,
detail::float2half<half::round_style>(detail::half2float<detail::internal_t>(x.data_)+detail::half2float<detail::intern
02556         #else
02557             int absx = x.data_ & 0x7FFF, absy = y.data_ & 0x7FFF;
02558             bool sub = ((x.data_^y.data_)&0x8000) != 0;
02559             if(absx >= 0x7C00 || absy >= 0x7C00)
02560                 return half(detail::binary, (absx>0x7C00 || absy>0x7C00) ? detail::signal(x.data_,
y.data_) : (absy!=0x7C00) ? x.data_ :
02561                     (sub && absx==0x7C00) ? detail::invalid() : y.data_);
02562             if(!absx)
02563                 return absy ? y : half(detail::binary, (half::round_style==std::round_toward_neg_infinity)
? (x.data_|y.data_) : (x.data_&y.data_));
02564             if(!absy)
02565                 return x;
02566             unsigned int sign = ((sub && absy>absx) ? y.data_ : x.data_) & 0x8000;
02567             if(absy > absx)
02568                 std::swap(absx, absy);
02569             int exp = (absx>10) + (absx<=0x3FF), d = exp - (absy>10) - (absy<=0x3FF), mx =
((absx<0x3FF) || ((absx>0x3FF)<10)) << 3, my;
02570             if(d < 13)
02571             {
02572                 my = ((absy<0x3FF) || ((absy>0x3FF)<10)) << 3;
02573                 my = (my>d) | ((my&((1<d)-1))!=0);
02574             }
02575             else

```



```

02576         my = 1;
02577         if(sub)
02578         {
02579             if(!(mx==my))
02580                 return half(detail::binary,
static_cast<unsigned>(half::round_style==std::round_toward_neg_infinity)<15>;
02581                 for(; mx<0x2000 && exp>1; mx<=1,--exp) ;
02582             }
02583             else
02584             {
02585                 mx += my;
02586                 int i = mx >> 14;
02587                 if((exp+=i) > 30)
02588                     return half(detail::binary, detail::overflow<half::round_style>(sign));
02589                 mx = (mx>>i) | (mx&i);
02590             }
02591             return half(detail::binary, detail::rounded<half::round_style,false>(sign+((exp-1)<10)+(mx>>3),
(mx>>2)&1, (mx&0x3)!=0));
02592         }
02593     }
02594
02602     inline half operator-(half x, half y)
02603     {
02604         #ifdef HALF_ARITHMETIC_TYPE
02605             return half(detail::binary,
detail::float2half<half::round_style>(detail::half2float<detail::internal_t>(x.data_)-detail::half2float<detail::intern
02606             #else
02607                 return x + -y;
02608             #endif
02609         }
02610
02618     inline half operator*(half x, half y)
02619     {
02620         #ifdef HALF_ARITHMETIC_TYPE
02621             return half(detail::binary,
detail::float2half<half::round_style>(detail::half2float<detail::internal_t>(x.data_)*detail::half2float<detail::intern
02622             #else
02623                 int absx = x.data_ & 0x7FFF, absy = y.data_ & 0x7FFF, exp = -16;
02624                 unsigned int sign = (x.data_^y.data_) & 0x8000;
02625                 if(absx >= 0x7C00 || absy >= 0x7C00)
02626                     return half(detail::binary, (absx>0x7C00 || absy>0x7C00) ? detail::signal(x.data_,
y.data_) :
02627                                     ((absx==0x7C00 && !absy)|| (absy==0x7C00 && !absx)) ?
detail::invalid() : (sign|0x7C00));
02628                 if(!absx || !absy)
02629                     return half(detail::binary, sign);
02630                 for(; absx<0x400; absx<=1,--exp) ;
02631                 for(; absy<0x400; absy<=1,--exp) ;
02632                 detail::uint32 m = static_cast<detail::uint32>((absx&0x3FF)|0x400) *
static_cast<detail::uint32>((absy&0x3FF)|0x400);
02633                 int i = m >> 21, s = m & i;
02634                 exp += (absx>>10) + (absy>>10) + i;
02635                 if(exp > 29)
02636                     return half(detail::binary, detail::overflow<half::round_style>(sign));
02637                 else if(exp < -11)
02638                     return half(detail::binary, detail::underflow<half::round_style>(sign));
02639                 return half(detail::binary, detail::fixed2half<half::round_style,20,false,false>(mx>i,
exp, sign, s));
02640             #endif
02641         }
02642
02651     inline half operator/(half x, half y)
02652     {
02653         #ifdef HALF_ARITHMETIC_TYPE
02654             return half(detail::binary,
detail::float2half<half::round_style>(detail::half2float<detail::internal_t>(x.data_)/detail::half2float<detail::intern
02655             #else
02656                 int absx = x.data_ & 0x7FFF, absy = y.data_ & 0x7FFF, exp = 14;
02657                 unsigned int sign = (x.data_^y.data_) & 0x8000;
02658                 if(absx >= 0x7C00 || absy >= 0x7C00)
02659                     return half(detail::binary, (absx>0x7C00 || absy>0x7C00) ? detail::signal(x.data_,
y.data_) :
02660                                     (absx==absy) ? detail::invalid() : (sign|((absx==0x7C00) ?
0x7C00 : 0)));
02661                 if(!absx)
02662                     return half(detail::binary, absy ? sign : detail::invalid());
02663                 if(!absy)
02664                     return half(detail::binary, detail::pole(sign));
02665                 for(; absx<0x400; absx<=1,--exp) ;
02666                 for(; absy<0x400; absy<=1,++exp) ;
02667                 detail::uint32 mx = (absx&0x3FF) | 0x400, my = (absy&0x3FF) | 0x400;
02668                 int i = mx < my;
02669                 exp += (absx>>10) - (absy>>10) - i;
02670                 if(exp > 29)
02671                     return half(detail::binary, detail::overflow<half::round_style>(sign));
02672                 else if(exp < -11)
02673                     return half(detail::binary, detail::underflow<half::round_style>(sign));

```

```

02674         mx <= 12 + i;
02675         my <= 1;
02676         return half(detail::binary, detail::fixed2half<half::round_style,11,false,false,false>(mx/my,
exp, sign, mx%my!=0));
02677     #endif
02678 }
02679
02684
02690     template<typename charT,typename traits> std::basic_ostream<charT,traits>&
operator<<(std::basic_ostream<charT,traits> &out, half arg)
02691     {
02692     #ifdef HALF_ARITHMETIC_TYPE
02693         return out << detail::half2float<detail::internal_t>(arg.data_);
02694     #else
02695         return out << detail::half2float<float>(arg.data_);
02696     #endif
02697     }
02698
02708     template<typename charT,typename traits> std::basic_istream<charT,traits>&
operator>>(std::basic_istream<charT,traits> &in, half &arg)
02709     {
02710     #ifdef HALF_ARITHMETIC_TYPE
02711         detail::internal_t f;
02712     #else
02713         double f;
02714     #endif
02715         if(in >> f)
02716             arg.data_ = detail::float2half<half::round_style>(f);
02717         return in;
02718     }
02719
02724
02729     inline HALF_CONSTEXPR half fabs(half arg) { return half(detail::binary, arg.data_&0xFFFF); }
02730
02735     inline HALF_CONSTEXPR half abs(half arg) { return fabs(arg); }
02736
02743     inline half fmod(half x, half y)
02744     {
02745         unsigned int absx = x.data_ & 0xFFFF, absy = y.data_ & 0xFFFF, sign = x.data_ & 0x8000;
02746         if(absx >= 0x7C00 || absy >= 0x7C00)
02747             return half(detail::binary, (absx>0x7C00 || absy>0x7C00) ? detail::signal(x.data_,
y.data_) :
02748                 (absx==0x7C00) ? detail::invalid() : x.data_);
02749         if(!absy)
02750             return half(detail::binary, detail::invalid());
02751         if(!absx)
02752             return x;
02753         if(absx == absy)
02754             return half(detail::binary, sign);
02755         return half(detail::binary, sign|detail::mod<false,false>(absx, absy));
02756     }
02757
02764     inline half remainder(half x, half y)
02765     {
02766         unsigned int absx = x.data_ & 0xFFFF, absy = y.data_ & 0xFFFF, sign = x.data_ & 0x8000;
02767         if(absx >= 0x7C00 || absy >= 0x7C00)
02768             return half(detail::binary, (absx>0x7C00 || absy>0x7C00) ? detail::signal(x.data_,
y.data_) :
02769                 (absx==0x7C00) ? detail::invalid() : x.data_);
02770         if(!absy)
02771             return half(detail::binary, detail::invalid());
02772         if(absx == absy)
02773             return half(detail::binary, sign);
02774         return half(detail::binary, sign^detail::mod<false,true>(absx, absy));
02775     }
02776
02784     inline half remquo(half x, half y, int *quo)
02785     {
02786         unsigned int absx = x.data_ & 0xFFFF, absy = y.data_ & 0xFFFF, value = x.data_ & 0x8000;
02787         if(absx >= 0x7C00 || absy >= 0x7C00)
02788             return half(detail::binary, (absx>0x7C00 || absy>0x7C00) ? detail::signal(x.data_,
y.data_) :
02789                 (absx==0x7C00) ? detail::invalid() : (*quo = 0, x.data_));
02790         if(!absy)
02791             return half(detail::binary, detail::invalid());
02792         bool qsign = ((value^y.data_)&0x8000) != 0;
02793         int q = 1;
02794         if(absx != absy)
02795             value ^= detail::mod<true, true>(absx, absy, &q);
02796         return *quo = qsign ? -q : q, half(detail::binary, value);
02797     }
02798
02809     inline half fma(half x, half y, half z)
02810     {
02811     #ifdef HALF_ARITHMETIC_TYPE
02812         detail::internal_t fx = detail::half2float<detail::internal_t>(x.data_), fy =
detail::half2float<detail::internal_t>(y.data_), fz = detail::half2float<detail::internal_t>(z.data_);

```

```

02813         #if HALF_ENABLE_CPP11_CMATH && FP_FAST_FMA
02814             return half(detail::binary, detail::float2half<half::round_style>(std::fma(fx, fy, fz)));
02815         #else
02816             return half(detail::binary, detail::float2half<half::round_style>(fx*fy+fz));
02817         #endif
02818     #else
02819         int absx = x.data_ & 0x7FFF, absy = y.data_ & 0x7FFF, absz = z.data_ & 0x7FFF, exp = -15;
02820         unsigned int sign = (x.data_ ^ y.data_) & 0x8000;
02821         bool sub = ((sign ^ z.data_) & 0x8000) != 0;
02822         if(absx >= 0x7C00 || absy >= 0x7C00 || absz >= 0x7C00)
02823             return (absx > 0x7C00 || absy > 0x7C00 || absz > 0x7C00) ? half(detail::binary,
detail::signal(x.data_, y.data_, z.data_)) :
02824             (absx == 0x7C00) ? half(detail::binary, (!absy || (sub && absz == 0x7C00)) ?
detail::invalid() : (sign | 0x7C00)) :
02825             (absy == 0x7C00) ? half(detail::binary, (!absx || (sub && absz == 0x7C00)) ?
detail::invalid() : (sign | 0x7C00)) : z;
02826         if(!absx || !absy)
02827             return absz ? z : half(detail::binary, (half::round_style == std::round_toward_neg_infinity)
? (z.data_ | sign) : (z.data_ & sign));
02828         for(; absx < 0x400; absx <= 1, --exp);
02829         for(; absy < 0x400; absy <= 1, --exp);
02830         detail::uint32 m = static_cast<detail::uint32>((absx & 0x3FF) | 0x400) *
static_cast<detail::uint32>((absy & 0x3FF) | 0x400);
02831         int i = m >> 21;
02832         exp += (absx >> 10) + (absy >> 10) + i;
02833         m <<= 3 - i;
02834         if(absz)
02835         {
02836             int expz = 0;
02837             for(; absz < 0x400; absz <= 1, --expz);
02838             expz += absz >> 10;
02839             detail::uint32 mz = static_cast<detail::uint32>((absz & 0x3FF) | 0x400) << 13;
02840             if(expz > exp || (expz == exp && mz > m))
02841             {
02842                 std::swap(m, mz);
02843                 std::swap(exp, expz);
02844                 if(sub)
02845                     sign = z.data_ & 0x8000;
02846             }
02847             int d = exp - expz;
02848             mz = (d < 23) ? ((mz >> d) | ((mz & ((static_cast<detail::uint32>(1) << d) - 1)) != 0)) : 1;
02849             if(sub)
02850             {
02851                 m = m - mz;
02852                 if(!m)
02853                     return half(detail::binary,
static_cast<unsigned>(half::round_style == std::round_toward_neg_infinity) << 15);
02854                 for(; m < 0x800000; m <= 1, --exp);
02855             }
02856             else
02857             {
02858                 m += mz;
02859                 i = m >> 24;
02860                 m = (m >> i) | (m & i);
02861                 exp += i;
02862             }
02863         }
02864         if(exp > 30)
02865             return half(detail::binary, detail::overflow<half::round_style>(sign));
02866         else if(exp < -10)
02867             return half(detail::binary, detail::underflow<half::round_style>(sign));
02868         return half(detail::binary, detail::fixed2half<half::round_style, 23, false, false, false>(m,
exp-1, sign));
02869     #endif
02870 }
02871
02872 inline HALF_CONSTEXPR_NOERR half fmax(half x, half y)
02873 {
02874     return half(detail::binary, (!isnan(y) && (isnan(x) ||
(x.data_ ^ (0x8000 | (0x8000 - (x.data_ >> 15)))) <
(y.data_ ^ (0x8000 | (0x8000 - (y.data_ >> 15)))))) ? detail::select(y.data_, x.data_) :
detail::select(x.data_, y.data_));
02875 }
02876
02877 inline HALF_CONSTEXPR_NOERR half fmin(half x, half y)
02878 {
02879     return half(detail::binary, (!isnan(y) && (isnan(x) ||
(x.data_ ^ (0x8000 | (0x8000 - (x.data_ >> 15)))) >
(y.data_ ^ (0x8000 | (0x8000 - (y.data_ >> 15)))))) ? detail::select(y.data_, x.data_) :
detail::select(x.data_, y.data_));
02880 }
02881
02882 inline half fdim(half x, half y)
02883 {
02884     if(isnan(x) || isnan(y))
02885         return half(detail::binary, detail::signal(x.data_, y.data_));
02886     return (x.data_ ^ (0x8000 | (0x8000 - (x.data_ >> 15)))) <= (y.data_ ^ (0x8000 | (0x8000 - (y.data_ >> 15)))) ?

```

```

    half(detail::binary, 0) : (x-y);
02909     }
02910
02915     inline half nanh(const char *arg)
02916     {
02917         unsigned int value = 0x7FFF;
02918         while(*arg)
02919             value ^= static_cast<unsigned>(*arg++) & 0xFF;
02920         return half(detail::binary, value);
02921     }
02922
02927
02936     inline half exp(half arg)
02937     {
02938         #ifdef HALF_ARITHMETIC_TYPE
02939             return half(detail::binary,
02940 detail::float2half<half::round_style>(std::exp(detail::half2float<detail::internal_t>(arg.data_))));
02941         #else
02942             int abs = arg.data_ & 0x7FFF, e = (abs>>10) + (abs<=0x3FF), exp;
02943             if(!abs)
02944                 return half(detail::binary, 0x3C00);
02945             if(abs >= 0x7C00)
02946                 return half(detail::binary, (abs==0x7C00) ? (0x7C00+((arg.data_>>15)-1U)) :
02947 detail::signal(arg.data_));
02948             if(abs >= 0x4C80)
02949                 return half(detail::binary, (arg.data_&0x8000) ? detail::underflow<half::round_style>() :
02950 detail::overflow<half::round_style>());
02951             detail::uint32 m =
02952 detail::multiply64(static_cast<detail::uint32>((abs<0x3FF)+(abs>0x3FF)<<10))<<21, 0xB8AA3B29);
02953             if(e < 14)
02954             {
02955                 exp = 0;
02956                 m >>= 14 - e;
02957             }
02958             else
02959             {
02960                 exp = m >> (45-e);
02961                 m = (m<<(e-14)) & 0x7FFFFFFF;
02962             }
02963             return half(detail::binary, detail::exp2_post<half::round_style>(m, exp,
02964 (arg.data_&0x8000)!=0, 0, 26));
02965         #endif
02966     }
02967
02971     inline half exp2(half arg)
02972     {
02973         #if defined(HALF_ARITHMETIC_TYPE) && HALF_ENABLE_CPP11_CMATH
02974             return half(detail::binary,
02975 detail::float2half<half::round_style>(std::exp2(detail::half2float<detail::internal_t>(arg.data_))));
02976         #else
02977             int abs = arg.data_ & 0x7FFF, e = (abs>>10) + (abs<=0x3FF), exp = (abs<0x3FF) +
02978 ((abs>0x3FF)<<10);
02979             if(!abs)
02980                 return half(detail::binary, 0x3C00);
02981             if(abs >= 0x7C00)
02982                 return half(detail::binary, (abs==0x7C00) ? (0x7C00+((arg.data_>>15)-1U)) :
02983 detail::signal(arg.data_));
02984             if(abs >= 0x4E40)
02985                 return half(detail::binary, (arg.data_&0x8000) ? detail::underflow<half::round_style>() :
02986 detail::overflow<half::round_style>());
02987             return half(detail::binary, detail::exp2_post<half::round_style>(
02988 (static_cast<detail::uint32>(exp)<<(6+e))&0x7FFFFFFF, exp>>(25-e), (arg.data_&0x8000)!=0, 0,
02989 28));
02990         #endif
02991     }
02992
02997     inline half expm1(half arg)
02998     {
02999         #if defined(HALF_ARITHMETIC_TYPE) && HALF_ENABLE_CPP11_CMATH
03000             return half(detail::binary,
03001 detail::float2half<half::round_style>(std::expm1(detail::half2float<detail::internal_t>(arg.data_))));
03002         #else
03003             unsigned int abs = arg.data_ & 0x7FFF, sign = arg.data_ & 0x8000, e = (abs>>10) + (abs<=0x3FF),
03004 exp;
03005             if(!abs)
03006                 return arg;
03007             if(abs >= 0x7C00)
03008                 return half(detail::binary, (abs==0x7C00) ? (0x7C00+(sign>>1)) :
03009 detail::signal(arg.data_));
03010             if(abs >= 0x4A00)
03011                 return half(detail::binary, (arg.data_&0x8000) ?
03012 detail::rounded<half::round_style,true>(0xBBFF, 1, 1) : detail::overflow<half::round_style>());
03013             detail::uint32 m =
03014 detail::multiply64(static_cast<detail::uint32>((abs<0x3FF)+(abs>0x3FF)<<10))<<21, 0xB8AA3B29);
03015             if(e < 14)
03016             {
03017                 exp = 0;

```

```

03013         m >= 14 - e;
03014     }
03015     else
03016     {
03017         exp = m > (45-e);
03018         m = (m<<(e-14)) & 0x7FFFFFFF;
03019     }
03020     m = detail::exp2(m);
03021     if(sign)
03022     {
03023         int s = 0;
03024         if(m > 0x80000000)
03025         {
03026             ++exp;
03027             m = detail::divide64(0x80000000, m, s);
03028         }
03029         m = 0x80000000 - ((m>exp) | ((m<((static_cast<detail::uint32>(1)<<exp)-1)) != 0) | s);
03030         exp = 0;
03031     }
03032     else
03033     {
03034         m -= (exp<31) ? (0x80000000<<exp) : 1;
03035         for(exp+=14; m<0x80000000 && exp; m<=1,--exp) ;
03036         if(exp > 29)
03037             return half(detail::binary, detail::overflow<half::round_style>());
03038         return half(detail::binary, detail::rounded<half::round_style,true>(sign+(exp<10)+(m>21),
03039             (m>20)&1, (m&0xFFFF) != 0));
03040     }
03041 }
03042
03043 inline half log(half arg)
03044 {
03045     #ifdef HALF_ARITHMETIC_TYPE
03046         return half(detail::binary,
03047             detail::float2half<half::round_style>(std::log(detail::half2float<detail::internal_t>(arg.data_))));
03048     #else
03049         int abs = arg.data_ & 0x7FFF, exp = -15;
03050         if(!abs)
03051             return half(detail::binary, detail::pole(0x8000));
03052         if(arg.data_ & 0x8000)
03053             return half(detail::binary, (arg.data_<=0xFC00) ? detail::invalid() :
03054                 detail::signal(arg.data_));
03055         if(abs >= 0x7C00)
03056             return (abs==0x7C00) ? arg : half(detail::binary, detail::signal(arg.data_));
03057         for(; abs<0x400; abs<=1,--exp) ;
03058         exp += abs > 10;
03059         return half(detail::binary, detail::log2_post<half::round_style,0xB8AA3B2A>(
03060             detail::log2(static_cast<detail::uint32>((abs&0x3FF)|0x400)<<20, 27)+8, exp, 17));
03061     #endif
03062 }
03063
03064 inline half log10(half arg)
03065 {
03066     #ifdef HALF_ARITHMETIC_TYPE
03067         return half(detail::binary,
03068             detail::float2half<half::round_style>(std::log10(detail::half2float<detail::internal_t>(arg.data_))));
03069     #else
03070         int abs = arg.data_ & 0x7FFF, exp = -15;
03071         if(!abs)
03072             return half(detail::binary, detail::pole(0x8000));
03073         if(arg.data_ & 0x8000)
03074             return half(detail::binary, (arg.data_<=0xFC00) ? detail::invalid() :
03075                 detail::signal(arg.data_));
03076         if(abs >= 0x7C00)
03077             return (abs==0x7C00) ? arg : half(detail::binary, detail::signal(arg.data_));
03078         switch(abs)
03079         {
03080             case 0x4900: return half(detail::binary, 0x3C00);
03081             case 0x5640: return half(detail::binary, 0x4000);
03082             case 0x63D0: return half(detail::binary, 0x4200);
03083             case 0x70E2: return half(detail::binary, 0x4400);
03084         }
03085         for(; abs<0x400; abs<=1,--exp) ;
03086         exp += abs > 10;
03087         return half(detail::binary, detail::log2_post<half::round_style,0xD49A784C>(
03088             detail::log2(static_cast<detail::uint32>((abs&0x3FF)|0x400)<<20, 27)+8, exp, 16));
03089     #endif
03090 }
03091
03092 inline half log2(half arg)
03093 {
03094     #if defined(HALF_ARITHMETIC_TYPE) && HALF_ENABLE_CPP11_CMATH
03095         return half(detail::binary,
03096             detail::float2half<half::round_style>(std::log2(detail::half2float<detail::internal_t>(arg.data_))));
03097     #else
03098         int abs = arg.data_ & 0x7FFF, exp = -15, s = 0;
03099         if(!abs)
03100             return half(detail::binary, detail::pole(0x8000));
03101     #endif

```

```

03121         if(arg.data_ & 0x8000)
03122             return half(detail::binary, (arg.data_<=0xFC00) ? detail::invalid() :
detail::signal(arg.data_));
03123         if(abs >= 0x7C00)
03124             return (abs==0x7C00) ? arg : half(detail::binary, detail::signal(arg.data_));
03125         if(abs == 0x3C00)
03126             return half(detail::binary, 0);
03127         for(; abs<0x400; abs<=1,--exp) ;
03128         exp += (abs>10);
03129         if(!(abs&0x3FF))
03130         {
03131             unsigned int value = static_cast<unsigned>(exp<0) << 15, m = std::abs(exp) << 6;
03132             for(exp=18; m<0x400; m<=1,--exp) ;
03133             return half(detail::binary, value+(exp<10)+m);
03134         }
03135         detail::uint32 ilog = exp, sign = detail::sign_mask(ilog), m =
03136             ((ilog<27)+(detail::log2(static_cast<detail::uint32>((abs&0x3FF)|0x400)<<20, 28)>>4))^sign)
- sign;
03137         if(!m)
03138             return half(detail::binary, 0);
03139         for(exp=14; m<0x8000000 && exp; m<=1,--exp) ;
03140         for(; m>0xFFFFFFFF; m>=1,++exp)
03141             s |= m & 1;
03142         return half(detail::binary, detail::fixed2half<half::round_style,27,false,false,true>(m, exp,
sign&0x8000, s));
03143     #endif
03144 }
03145
03156 inline half loglp(half arg)
03157 {
03158     #if defined(HALF_ARITHMETIC_TYPE) && HALF_ENABLE_CPP11_CMATH
03159         return half(detail::binary,
detail::float2half<half::round_style>(std::loglp(detail::half2float<detail::internal_t>(arg.data_))));
03160     #else
03161         if(arg.data_ >= 0xBC00)
03162             return half(detail::binary, (arg.data_==0xBC00) ? detail::pole(0x8000) :
(arg.data_<=0xFC00) ? detail::invalid() : detail::signal(arg.data_));
03163         int abs = arg.data_ & 0x7FFF, exp = -15;
03164         if(!abs || abs >= 0x7C00)
03165             return (abs>0x7C00) ? half(detail::binary, detail::signal(arg.data_)) : arg;
03166         for(; abs<0x400; abs<=1,--exp) ;
03167         exp += abs > 10;
03168         detail::uint32 m = static_cast<detail::uint32>((abs&0x3FF)|0x400) << 20;
03169         if(arg.data_ & 0x8000)
03170         {
03171             m = 0x40000000 - (m>>-exp);
03172             for(exp=0; m<0x40000000; m<=1,--exp) ;
03173         }
03174         else
03175         {
03176             if(exp < 0)
03177             {
03178                 m = 0x40000000 + (m>>-exp);
03179                 exp = 0;
03180             }
03181             else
03182             {
03183                 m += 0x40000000 >> exp;
03184                 int i = m >> 31;
03185                 m >>= i;
03186                 exp += i;
03187             }
03188         }
03189         return half(detail::binary, detail::log2_post<half::round_style,0xB8AA3B2A>(detail::log2(m),
exp, 17));
03190     #endif
03191 }
03192
03197 inline half sqrt(half arg)
03200 {
03201     #ifdef HALF_ARITHMETIC_TYPE
03202         return half(detail::binary,
detail::float2half<half::round_style>(std::sqrt(detail::half2float<detail::internal_t>(arg.data_))));
03203     #else
03204         int abs = arg.data_ & 0x7FFF, exp = 15;
03205         if(!abs || arg.data_ >= 0x7C00)
03206             return half(detail::binary, (abs>0x7C00) ? detail::signal(arg.data_) : (arg.data_>0x8000)
? detail::invalid() : arg.data_);
03207         for(; abs<0x400; abs<=1,--exp) ;
03208         detail::uint32 r = static_cast<detail::uint32>((abs&0x3FF)|0x400) << 10, m =
detail::sqrt<20>(r, exp+=abs>10);
03209         return half(detail::binary, detail::rounded<half::round_style,false>((exp<10)+(m&0x3FF), r>m,
r!=0));
03210     #endif
03211 }
03212
03213
03214
03215
03216
03217
03218
03219

```

```

03227     inline half rsqrt(half arg)
03228     {
03229         #ifdef HALF_ARITHMETIC_TYPE
03230             return half(detail::binary,
03231                 detail::float2half<half::round_style>(detail::internal_t(1)/std::sqrt(detail::half2float<detail::internal_t>(arg.data_)
03232                 #else
03233                 unsigned int abs = arg.data_ & 0x7FFF, bias = 0x4000;
03234                 if(!abs || arg.data_ >= 0x7C00)
03235                     return half(detail::binary, (abs>0x7C00) ? detail::signal(arg.data_) :
03236                     (arg.data_>0x8000) ?
03237                     detail::invalid() : !abs ? detail::pole(arg.data_&0x8000) :
03238                     0);
03239                 for(; abs<0x400; abs<=1,bias-=0x400) ;
03240                 unsigned int frac = (abs+=bias) & 0x7FFF;
03241                 if(frac == 0x400)
03242                     return half(detail::binary, 0x7A00-(abs>1));
03243                 if((half::round_style == std::round_to_nearest && (frac == 0x3FE || frac == 0x76C)) ||
03244                     (half::round_style != std::round_to_nearest && (frac == 0x15A || frac == 0x3FC || frac ==
03245                     0x401 || frac == 0x402 || frac == 0x67B)))
03246                     return pow(arg, half(detail::binary, 0xB800));
03247                 detail::uint32 f = 0x17376 - abs, mx = (abs&0x3FF) | 0x400, my = ((f>1)&0x3FF) | 0x400, mz =
03248                 my * my;
03249                 int expy = (f>1) - 31, expx = 32 - (abs>10), i = mz >> 21;
03250                 for(mz=0x60000000-(((mz>1)*mx)>>(expx-2*expy-i)); mz<0x40000000; mz<=1,--expy) ;
03251                 i = (my*=mz>10) >> 31;
03252                 expy += i;
03253                 my = (my>>(20+i)) + 1;
03254                 i = (mz=my*my) >> 21;
03255                 for(mz=0x60000000-(((mz>1)*mx)>>(expx-2*expy-i)); mz<0x40000000; mz<=1,--expy) ;
03256                 i = (my*=mz>10)+1 >> 31;
03257                 return half(detail::binary, detail::fixed2half<half::round_style,30,false,false,true>(my>1,
03258                 expy+i+14));
03259             #endif
03260         }
03261     }
03262     inline half cbrt(half arg)
03263     {
03264         #if defined(HALF_ARITHMETIC_TYPE) && HALF_ENABLE_CPP11_CMATH
03265             return half(detail::binary,
03266                 detail::float2half<half::round_style>(std::cbrt(detail::half2float<detail::internal_t>(arg.data_))));
03267         #else
03268             int abs = arg.data_ & 0x7FFF, exp = -15;
03269             if(!abs || abs == 0x3C00 || abs >= 0x7C00)
03270                 return (abs>0x7C00) ? half(detail::binary, detail::signal(arg.data_)) : arg;
03271             for(; abs<0x400; abs<=1, --exp);
03272             detail::uint32 ilog = exp + (abs>10), sign = detail::sign_mask(ilog), f, m =
03273             (((ilog>27)+(detail::log2(static_cast<detail::uint32>((abs&0x3FF)|0x400)<<20, 24)>>4))^sign)
03274             - sign;
03275             for(exp=2; m<0x80000000; m<=1,--exp) ;
03276             m = detail::multiply64(m, 0xAAAAAAB);
03277             int i = m >> 31, s;
03278             exp += i;
03279             m <= 1 - i;
03280             if(exp < 0)
03281             {
03282                 f = m >> -exp;
03283                 exp = 0;
03284             }
03285             else
03286             {
03287                 f = (m<<exp) & 0x7FFFFFFF;
03288                 exp = m >> (31-exp);
03289             }
03290             m = detail::exp2(f, (half::round_style==std::round_to_nearest) ? 29 : 26);
03291             if(sign)
03292             {
03293                 if(m > 0x80000000)
03294                 {
03295                     m = detail::divide64(0x80000000, m, s);
03296                     ++exp;
03297                 }
03298                 exp = -exp;
03299             }
03300             return half(detail::binary, (half::round_style==std::round_to_nearest) ?
03301                 detail::fixed2half<half::round_style,31,false,false,false>(m, exp+14, arg.data_&0x8000) :
03302                 detail::fixed2half<half::round_style,23,false,false,false>((m+0x80)>>8, exp+14,
03303                 arg.data_&0x8000));
03304             #endif
03305         }
03306     }
03307     inline half hypot(half x, half y)
03308     {
03309         #ifdef HALF_ARITHMETIC_TYPE
03310             detail::internal_t fx = detail::half2float<detail::internal_t>(x.data_), fy =
03311             detail::half2float<detail::internal_t>(y.data_);
03312             #if HALF_ENABLE_CPP11_CMATH
03313                 return half(detail::binary, detail::float2half<half::round_style>(std::hypot(fx, fy)));
03314             #endif
03315         }
03316     }

```

```

03321         #else
03322             return half(detail::binary,
detail::float2half<half::round_style>(std::sqrt(fx*fx+fy*fy)));
03323         #endif
03324     #else
03325         int absx = x.data_ & 0x7FFF, absy = y.data_ & 0x7FFF, expx = 0, expy = 0;
03326         if(absx >= 0x7C00 || absy >= 0x7C00)
03327             return half(detail::binary, (absx==0x7C00) ? detail::select(0x7C00, y.data_) :
03328                 (absy==0x7C00) ? detail::select(0x7C00, x.data_) : detail::signal(x.data_, y.data_));
03329         if(!absx)
03330             return half(detail::binary, absy ? detail::check_underflow(absy) : 0);
03331         if(!absy)
03332             return half(detail::binary, detail::check_underflow(absx));
03333         if(absx > absy)
03334             std::swap(absx, absy);
03335         for(; absx<0x400; absx<=1,--expx) ;
03336         for(; absy<0x400; absy<=1,--expy) ;
03337         detail::uint32 mx = (absx&0x3FF) | 0x400, my = (absy&0x3FF) | 0x400;
03338         mx *= mx;
03339         my *= my;
03340         int ix = mx >> 21, iy = my >> 21;
03341         expx = 2*(expx+(absx>>10)) - 15 + ix;
03342         expy = 2*(expy+(absy>>10)) - 15 + iy;
03343         mx <= 10 - ix;
03344         my <= 10 - iy;
03345         int d = expx - expy;
03346         my = (d<30) ? ((my>>d) | ((my&((static_cast<detail::uint32>(1)<<d)-1))!=0)) : 1;
03347         return half(detail::binary, detail::hypot_post<half::round_style>(mx+my, expx));
03348     #endif
03349 }
03350
03361 inline half hypot(half x, half y, half z)
03362 {
03363     #ifdef HALF_ARITHMETIC_TYPE
03364         detail::internal_t fx = detail::half2float<detail::internal_t>(x.data_), fy =
detail::half2float<detail::internal_t>(y.data_), fz = detail::half2float<detail::internal_t>(z.data_);
03365         return half(detail::binary,
detail::float2half<half::round_style>(std::sqrt(fx*fx+fy*fy+fz*fz)));
03366     #else
03367         int absx = x.data_ & 0x7FFF, absy = y.data_ & 0x7FFF, absz = z.data_ & 0x7FFF, expx = 0, expy
= 0, expz = 0;
03368         if(!absx)
03369             return hypot(y, z);
03370         if(!absy)
03371             return hypot(x, z);
03372         if(!absz)
03373             return hypot(x, y);
03374         if(absx >= 0x7C00 || absy >= 0x7C00 || absz >= 0x7C00)
03375             return half(detail::binary, (absx==0x7C00) ? detail::select(0x7C00,
detail::select(y.data_, z.data_)) :
03376                 (absy==0x7C00) ? detail::select(0x7C00,
detail::select(x.data_, z.data_)) :
03377                 (absz==0x7C00) ? detail::select(0x7C00,
detail::select(x.data_, y.data_)) :
03378                 detail::signal(x.data_, y.data_, z.data_));
03379         if(absz > absy)
03380             std::swap(absy, absz);
03381         if(absy > absx)
03382             std::swap(absx, absy);
03383         if(absz > absy)
03384             std::swap(absy, absz);
03385         for(; absx<0x400; absx<=1,--expx) ;
03386         for(; absy<0x400; absy<=1,--expy) ;
03387         for(; absz<0x400; absz<=1,--expz) ;
03388         detail::uint32 mx = (absx&0x3FF) | 0x400, my = (absy&0x3FF) | 0x400, mz = (absz&0x3FF) |
0x400;
03389         mx *= mx;
03390         my *= my;
03391         mz *= mz;
03392         int ix = mx >> 21, iy = my >> 21, iz = mz >> 21;
03393         expx = 2*(expx+(absx>>10)) - 15 + ix;
03394         expy = 2*(expy+(absy>>10)) - 15 + iy;
03395         expz = 2*(expz+(absz>>10)) - 15 + iz;
03396         mx <= 10 - ix;
03397         my <= 10 - iy;
03398         mz <= 10 - iz;
03399         int d = expy - expz;
03400         mz = (d<30) ? ((mz>>d) | ((mz&((static_cast<detail::uint32>(1)<<d)-1))!=0)) : 1;
03401         my += mz;
03402         if(my & 0x80000000)
03403         {
03404             my = (my>>1) | (my&1);
03405             if(++expy > expx)
03406             {
03407                 std::swap(mx, my);
03408                 std::swap(expx, expy);
03409             }

```



```

03410     }
03411     d = expx - expy;
03412     my = (d<30) ? ((my>d) | ((my&((static_cast<detail::uint32>(1)<<d)-1))!=0)) : 1;
03413     return half(detail::binary, detail::hypot_post<half::round_style>(mx+my, expx));
03414 #endif
03415 }
03416
03427 inline half pow(half x, half y)
03428 {
03429 #ifdef HALF_ARITHMETIC_TYPE
03430     return half(detail::binary,
03431 detail::float2half<half::round_style>(std::pow(detail::half2float<detail::internal_t>(x.data_),
03432 detail::half2float<detail::internal_t>(y.data_))));
03433 #else
03432     int absx = x.data_ & 0x7FFF, absy = y.data_ & 0x7FFF, exp = -15;
03433     if(!absy || x.data_ == 0x3C00)
03434         return half(detail::binary, detail::select(0x3C00, (x.data_==0x3C00) ? y.data_ :
03435 x.data_));
03436     bool is_int = absy >= 0x6400 || (absy==0x3C00 && !(absy&((1<<(25-(absy>10)))-1)));
03437     unsigned int sign = x.data_ &
03438 (static_cast<unsigned>((absy<0x6800)&&is_int&&((absy>(25-(absy>10))&1)<<15));
03439     if(absx >= 0x7C00 || absy >= 0x7C00)
03440         return half(detail::binary, (absx>0x7C00 || absy>0x7C00) ? detail::signal(x.data_,
03441 y.data_) :
03442 (absy==0x7C00) ? ((absx==0x3C00) ? 0x3C00 : (!absx &&
03443 y.data_==0xFC00) ? detail::pole() :
03444 (0x7C00&-(y.data_>15)^(absx>0x3C00)))) :
03445 (sign|(0x7C00&((y.data_>15)-1U))));
03446     if(!absx)
03447         return half(detail::binary, (y.data_&0x8000) ? detail::pole(sign) : sign);
03448     if((x.data_&0x8000) && !is_int)
03449         return half(detail::binary, detail::invalid());
03450     if(x.data_ == 0xBC00)
03451         return half(detail::binary, sign|0x3C00);
03452     switch(y.data_)
03453     {
03454     case 0x3800: return sqrt(x);
03455     case 0x3C00: return half(detail::binary, detail::check_underflow(x.data_));
03456     case 0x4000: return x * x;
03457     case 0xBC00: return half(detail::binary, 0x3C00) / x;
03458     }
03459     for(; absx<0x400; absx<=1,--exp) ;
03460     detail::uint32 ilog = exp + (absx>10), msign = detail::sign_mask(ilog), f, m =
03461 (((ilog<27)+((detail::log2(static_cast<detail::uint32>((absx&0x3FF)|0x400)<<20)+8)<<4))^msign) - msign);
03462     for(exp=-11; m<0x80000000; m<=1,--exp) ;
03463     for(; absy<0x400; absy<=1,--exp) ;
03464     m = detail::multiply64(m, static_cast<detail::uint32>((absy&0x3FF)|0x400)<<21);
03465     int i = m >> 31;
03466     exp += (absy>10) + i;
03467     m <= 1 - i;
03468     if(exp < 0)
03469     {
03470         f = m >> -exp;
03471         exp = 0;
03472     }
03473     else
03474     {
03475         f = (m<<exp) & 0x7FFFFFFF;
03476         exp = m >> (31-exp);
03477     }
03478     return half(detail::binary, detail::exp2_post<half::round_style>(f, exp,
03479 (msign&1)^(y.data_>15))!=0, sign));
03480 #endif
03481 }
03482
03491 inline void sincos(half arg, half *sin, half *cos)
03492 {
03493 #ifdef HALF_ARITHMETIC_TYPE
03494     detail::internal_t f = detail::half2float<detail::internal_t>(arg.data_);
03495     *sin = half(detail::binary, detail::float2half<half::round_style>(std::sin(f)));
03496     *cos = half(detail::binary, detail::float2half<half::round_style>(std::cos(f)));
03497 #else
03498     int abs = arg.data_ & 0x7FFF, sign = arg.data_ >> 15, k;
03499     if(abs >= 0x7C00)
03500         *sin = *cos = half(detail::binary, (abs==0x7C00) ? detail::invalid() :
03501 detail::signal(arg.data_));
03502     else if(!abs)
03503     {
03504         *sin = arg;
03505         *cos = half(detail::binary, 0x3C00);
03506     }
03507     else if(abs < 0x2500)
03508     {
03509         *sin = half(detail::binary, detail::rounded<half::round_style,true>(arg.data_-1, 1, 1));
03510         *cos = half(detail::binary, detail::rounded<half::round_style,true>(0x3BFF, 1, 1));
03511     }
03512     else
03513     {
03514         *sin = half(detail::binary, detail::rounded<half::round_style,true>(arg.data_-1, 1, 1));
03515         *cos = half(detail::binary, detail::rounded<half::round_style,true>(0x3BFF, 1, 1));
03516     }
03517 }

```

```

03510     }
03511     else
03512     {
03513         if(half::round_style != std::round_to_nearest)
03514         {
03515             switch(abs)
03516             {
03517                 case 0x48B7:
03518                     *sin = half(detail::binary,
03519 detail::rounded<half::round_style,true>((~arg.data_&0x8000)|0x1D07, 1, 1));
03519                     *cos = half(detail::binary, detail::rounded<half::round_style,true>(0xBBFF, 1,
1));
03520                     return;
03521                 case 0x598C:
03522                     *sin = half(detail::binary,
03523 detail::rounded<half::round_style,true>(arg.data_&0x8000)|0x3BFF, 1, 1));
03523                     *cos = half(detail::binary, detail::rounded<half::round_style,true>(0x80FC, 1,
1));
03524                     return;
03525                 case 0x6A64:
03526                     *sin = half(detail::binary,
03527 detail::rounded<half::round_style,true>((~arg.data_&0x8000)|0x3BFE, 1, 1));
03527                     *cos = half(detail::binary, detail::rounded<half::round_style,true>(0x27FF, 1,
1));
03528                     return;
03529                 case 0x6D8C:
03530                     *sin = half(detail::binary,
03531 detail::rounded<half::round_style,true>(arg.data_&0x8000)|0x0FE6, 1, 1));
03531                     *cos = half(detail::binary, detail::rounded<half::round_style,true>(0x3BFF, 1,
1));
03532                     return;
03533             }
03534         }
03535         std::pair<detail::uint32,detail::uint32> sc = detail::sincos(detail::angle_arg(abs, k),
28);
03536         switch(k & 3)
03537         {
03538             case 1: sc = std::make_pair(sc.second, -sc.first); break;
03539             case 2: sc = std::make_pair(-sc.first, -sc.second); break;
03540             case 3: sc = std::make_pair(-sc.second, sc.first); break;
03541         }
03542         *sin = half(detail::binary,
03543 detail::fixed2half<half::round_style,30,true,true,true>((sc.first^~static_cast<detail::uint32>(sign))+sign));
03543         *cos = half(detail::binary,
03544 detail::fixed2half<half::round_style,30,true,true,true>(sc.second));
03544     }
03545     #endif
03546 }
03547
03548 inline half sin(half arg)
03549 {
03550     #ifdef HALF_ARITHMETIC_TYPE
03551         return half(detail::binary,
03552 detail::float2half<half::round_style>(std::sin(detail::half2float<detail::internal_t>(arg.data_))));
03553     #else
03554         int abs = arg.data_ & 0x7FFF, k;
03555         if(!abs)
03556             return arg;
03557         if(abs >= 0x7C00)
03558             return half(detail::binary, (abs==0x7C00) ? detail::invalid() :
03559 detail::signal(arg.data_));
03560         if(abs < 0x2900)
03561             return half(detail::binary, detail::rounded<half::round_style,true>(arg.data_-1, 1, 1));
03562         if(half::round_style != std::round_to_nearest)
03563             switch(abs)
03564             {
03565                 case 0x48B7: return half(detail::binary,
03566 detail::rounded<half::round_style,true>((~arg.data_&0x8000)|0x1D07, 1, 1));
03567                 case 0x6A64: return half(detail::binary,
03568 detail::rounded<half::round_style,true>((~arg.data_&0x8000)|0x3BFE, 1, 1));
03569                 case 0x6D8C: return half(detail::binary,
03570 detail::rounded<half::round_style,true>(arg.data_&0x8000)|0x0FE6, 1, 1));
03571             }
03572         std::pair<detail::uint32,detail::uint32> sc = detail::sincos(detail::angle_arg(abs, k), 28);
03573         detail::uint32 sign = ~static_cast<detail::uint32>(((k>1)&1)^(arg.data_>15));
03574         return half(detail::binary, detail::fixed2half<half::round_style,30,true,true,true>(((k&1) ?
sc.second : sc.first)^sign) - sign));
03575     #endif
03576 }
03577
03578 inline half cos(half arg)
03579 {
03580     #ifdef HALF_ARITHMETIC_TYPE
03581         return half(detail::binary,
03582 detail::float2half<half::round_style>(std::cos(detail::half2float<detail::internal_t>(arg.data_))));
03583     #else
03584         int abs = arg.data_ & 0x7FFF, k;

```

```

03595         if(!abs)
03596             return half(detail::binary, 0x3C00);
03597         if(abs >= 0x7C00)
03598             return half(detail::binary, (abs==0x7C00) ? detail::invalid() :
detail::signal(arg.data_));
03599         if(abs < 0x2500)
03600             return half(detail::binary, detail::rounded<half::round_style,true>(0x3BFF, 1, 1));
03601         if(half::round_style != std::round_to_nearest && abs == 0x598C)
03602             return half(detail::binary, detail::rounded<half::round_style,true>(0x80FC, 1, 1));
03603         std::pair<detail::uint32,detail::uint32> sc = detail::sincos(detail::angle_arg(abs, k), 28);
03604         detail::uint32 sign = -static_cast<detail::uint32>(((k>1)^k)&1);
03605         return half(detail::binary, detail::fixed2half<half::round_style,30,true,true,true>(((k&1) ?
sc.first : sc.second)^sign) - sign));
03606     #endif
03607 }
03608
03617 inline half tan(half arg)
03618 {
03619     #ifdef HALF_ARITHMETIC_TYPE
03620         return half(detail::binary,
detail::float2half<half::round_style>(std::tan(detail::half2float<detail::internal_t>(arg.data_))));
03621     #else
03622         int abs = arg.data_ & 0x7FFF, exp = 13, k;
03623         if(!abs)
03624             return arg;
03625         if(abs >= 0x7C00)
03626             return half(detail::binary, (abs==0x7C00) ? detail::invalid() :
detail::signal(arg.data_));
03627         if(abs < 0x2700)
03628             return half(detail::binary, detail::rounded<half::round_style,true>(arg.data_, 0, 1));
03629         if(half::round_style != std::round_to_nearest)
03630             switch(abs)
03631             {
03632                 case 0x658C: return half(detail::binary,
detail::rounded<half::round_style,true>((arg.data_&0x8000)|0x07E6, 1, 1));
03633                 case 0x7330: return half(detail::binary,
detail::rounded<half::round_style,true>((~arg.data_&0x8000)|0x4B62, 1, 1));
03634             }
03635         std::pair<detail::uint32,detail::uint32> sc = detail::sincos(detail::angle_arg(abs, k), 30);
03636         if(k & 1)
03637             sc = std::make_pair(-sc.second, sc.first);
03638         detail::uint32 signy = detail::sign_mask(sc.first), signx = detail::sign_mask(sc.second);
03639         detail::uint32 my = (sc.first^signy) - signy, mx = (sc.second^signx) - signx;
03640         for(; my<0x80000000; my<=1,--exp) ;
03641         for(; mx<0x80000000; mx<=1,++exp) ;
03642         return half(detail::binary, detail::tangent_post<half::round_style>(my, mx, exp,
(signy^signx^arg.data_)&0x8000));
03643     #endif
03644 }
03645
03654 inline half asin(half arg)
03655 {
03656     #ifdef HALF_ARITHMETIC_TYPE
03657         return half(detail::binary,
detail::float2half<half::round_style>(std::asin(detail::half2float<detail::internal_t>(arg.data_))));
03658     #else
03659         unsigned int abs = arg.data_ & 0x7FFF, sign = arg.data_ & 0x8000;
03660         if(!abs)
03661             return arg;
03662         if(abs >= 0x3C00)
03663             return half(detail::binary, (abs>0x7C00) ? detail::signal(arg.data_) : (abs>0x3C00) ?
detail::invalid() :
detail::rounded<half::round_style,true>(sign|0x3E48, 0, 1));
03664         if(abs < 0x2900)
03665             return half(detail::binary, detail::rounded<half::round_style,true>(arg.data_, 0, 1));
03666         if(half::round_style != std::round_to_nearest && (abs == 0x2B44 || abs == 0x2DC3))
03667             return half(detail::binary, detail::rounded<half::round_style,true>(arg.data_+1, 1, 1));
03668         std::pair<detail::uint32,detail::uint32> sc = detail::atan2_args(abs);
03669         detail::uint32 m = detail::atan2(sc.first, sc.second,
(half::round_style==std::round_to_nearest) ? 27 : 26);
03670         return half(detail::binary, detail::fixed2half<half::round_style,30,false,true,true>(m, 14,
sign));
03671     #endif
03672 }
03673
03674 inline half acos(half arg)
03675 {
03676     #ifdef HALF_ARITHMETIC_TYPE
03677         return half(detail::binary,
detail::float2half<half::round_style>(std::acos(detail::half2float<detail::internal_t>(arg.data_))));
03678     #else
03679         unsigned int abs = arg.data_ & 0x7FFF, sign = arg.data_ >> 15;
03680         if(!abs)
03681             return half(detail::binary, detail::rounded<half::round_style,true>(0x3E48, 0, 1));
03682         if(abs >= 0x3C00)
03683             return half(detail::binary, (abs>0x7C00) ? detail::signal(arg.data_) : (abs>0x3C00) ?
detail::invalid() :

```

```

03693                                     sign ? detail::rounded<half::round_style,true>(0x4248, 0, 1) :
03694     0);
03694         std::pair<detail::uint32,detail::uint32> cs = detail::atan2_args(abs);
03695         detail::uint32 m = detail::atan2(cs.second, cs.first, 28);
03696         return half(detail::binary, detail::fixed2half<half::round_style,31,false,true,true>(sign ?
03697     (0xC90FDAA2-m) : m, 15, 0, sign));
03697     #endif
03698     }
03699
03700     inline half atan(half arg)
03701     {
03710     #ifdef HALF_ARITHMETIC_TYPE
03711         return half(detail::binary,
03712     detail::float2half<half::round_style>(std::atan(detail::half2float<detail::internal_t>(arg.data_))));
03712     #else
03713         unsigned int abs = arg.data_ & 0x7FFF, sign = arg.data_ & 0x8000;
03714         if(!abs)
03715             return arg;
03716         if(abs >= 0x7C00)
03717             return half(detail::binary, (abs==0x7C00) ?
03718     detail::rounded<half::round_style,true>(sign|0x3E48, 0, 1) : detail::signal(arg.data_));
03718         if(abs <= 0x2700)
03719             return half(detail::binary, detail::rounded<half::round_style,true>(arg.data_-1, 1, 1));
03720         int exp = (abs>>10) + (abs<=0x3FF);
03721         detail::uint32 my = (abs&0x3FF) | ((abs>0x3FF)<<10);
03722         detail::uint32 m = (exp>15) ? detail::atan2(my<<19, 0x20000000)<(exp-15),
03723     (half::round_style==std::round_to_nearest) ? 26 : 24) :
03724         detail::atan2(my<<(exp+4), 0x20000000,
03725     (half::round_style==std::round_to_nearest) ? 30 : 28);
03726         return half(detail::binary, detail::fixed2half<half::round_style,30,false,true,true>(m, 14,
03727     sign));
03728     #endif
03729     }
03730
03731     inline half atan2(half y, half x)
03732     {
03740     #ifdef HALF_ARITHMETIC_TYPE
03741         return half(detail::binary,
03742     detail::float2half<half::round_style>(std::atan2(detail::half2float<detail::internal_t>(y.data_),
03743     detail::half2float<detail::internal_t>(x.data_))));
03744     #else
03745         unsigned int absx = x.data_ & 0x7FFF, absy = y.data_ & 0x7FFF, signx = x.data_ >> 15, signy =
03746     y.data_ & 0x8000;
03747         if(absx >= 0x7C00 || absy >= 0x7C00)
03748         {
03749             if(absx > 0x7C00 || absy > 0x7C00)
03750                 return half(detail::binary, detail::signal(x.data_, y.data_));
03751             if(absy == 0x7C00)
03752                 return half(detail::binary, (absx<0x7C00) ?
03753     detail::rounded<half::round_style,true>(signy|0x3E48, 0, 1) :
03754                 signx ?
03755     detail::rounded<half::round_style,true>(signy|0x40B6, 0, 1) :
03756     detail::rounded<half::round_style,true>(signy|0x3A48, 0, 1));
03757             return (x.data_==0x7C00) ? half(detail::binary, signy) : half(detail::binary,
03758     detail::rounded<half::round_style,true>(signy|0x4248, 0, 1));
03759         }
03760         if(!absy)
03761             return signx ? half(detail::binary, detail::rounded<half::round_style,true>(signy|0x4248,
03762     0, 1)) : y;
03763         if(!absx)
03764             return half(detail::binary, detail::rounded<half::round_style,true>(signy|0x3E48, 0, 1));
03765         int d = (absy>>10) + (absy<=0x3FF) - (absx>>10) - (absx<=0x3FF);
03766         if(d > (signx ? 18 : 12))
03767             return half(detail::binary, detail::rounded<half::round_style,true>(signy|0x3E48, 0, 1));
03768         if(signx && d < -11)
03769             return half(detail::binary, detail::rounded<half::round_style,true>(signy|0x4248, 0, 1));
03770         if(!signx && d < ((half::round_style==std::round_toward_zero) ? -15 : -9))
03771         {
03772             for(; absy<0x400; absy<=1,--d) ;
03773             detail::uint32 mx = ((absx<<1)&0x7FF) | 0x800, my = ((absy<<1)&0x7FF) | 0x800;
03774             int i = my < mx;
03775             d -= i;
03776             if(d < -25)
03777                 return half(detail::binary, detail::underflow<half::round_style>(signy));
03778             my <<= 11 + i;
03779             return half(detail::binary,
03780     detail::fixed2half<half::round_style,11,false,false,true>(my/mx, d+14, signy, my%mx!=0));
03781         }
03782         detail::uint32 m = detail::atan2( ((absy<<0x3FF)|((absy>0x3FF)<<10))<<(19+((d<0) ? d : (d>0) ?
03783     0 : -1)),
03784         ((absx<<0x3FF)|((absx>0x3FF)<<10))<<(19-((d>0) ? d : (d<0) ?
03785     0 : 1)));
03786         return half(detail::binary, detail::fixed2half<half::round_style,31,false,true,true>(signx ?
03787     (0xC90FDAA2-m) : m, 15, signy, signx));
03788     #endif
03789     }

```

```

03779
03784
03793     inline half sinh(half arg)
03794     {
03795         #ifdef HALF_ARITHMETIC_TYPE
03796             return half(detail::binary,
detail::float2half<half::round_style>(std::sinh(detail::half2float<detail::internal_t>(arg.data_))));
03797         #else
03798             int abs = arg.data_ & 0x7FFF, exp;
03799             if(!abs || abs >= 0x7C00)
03800                 return (abs > 0x7C00) ? half(detail::binary, detail::signal(arg.data_)) : arg;
03801             if(abs <= 0x2900)
03802                 return half(detail::binary, detail::rounded<half::round_style,true>(arg.data_, 0, 1));
03803             std::pair<detail::uint32,detail::uint32> mm = detail::hyperbolic_args(abs, exp,
(half::round_style==std::round_to_nearest) ? 29 : 27);
03804             detail::uint32 m = mm.first - mm.second;
03805             for(exp+=13; m<0x80000000 && exp; m<=1,--exp) ;
03806             unsigned int sign = arg.data_ & 0x8000;
03807             if(exp > 29)
03808                 return half(detail::binary, detail::overflow<half::round_style>(sign));
03809             return half(detail::binary, detail::fixed2half<half::round_style,31,false,false,true>(m, exp,
sign));
03810         #endif
03811     }
03812
03821     inline half cosh(half arg)
03822     {
03823         #ifdef HALF_ARITHMETIC_TYPE
03824             return half(detail::binary,
detail::float2half<half::round_style>(std::cosh(detail::half2float<detail::internal_t>(arg.data_))));
03825         #else
03826             int abs = arg.data_ & 0x7FFF, exp;
03827             if(!abs)
03828                 return half(detail::binary, 0x3C00);
03829             if(abs >= 0x7C00)
03830                 return half(detail::binary, (abs > 0x7C00) ? detail::signal(arg.data_) : 0x7C00);
03831             std::pair<detail::uint32,detail::uint32> mm = detail::hyperbolic_args(abs, exp,
(half::round_style==std::round_to_nearest) ? 23 : 26);
03832             detail::uint32 m = mm.first + mm.second, i = (~m&0xFFFFFFFF) >> 31;
03833             m = (m>>i) | (m&i) | 0x80000000;
03834             if((exp+=13+i) > 29)
03835                 return half(detail::binary, detail::overflow<half::round_style>());
03836             return half(detail::binary, detail::fixed2half<half::round_style,31,false,false,true>(m,
exp));
03837         #endif
03838     }
03839
03848     inline half tanh(half arg)
03849     {
03850         #ifdef HALF_ARITHMETIC_TYPE
03851             return half(detail::binary,
detail::float2half<half::round_style>(std::tanh(detail::half2float<detail::internal_t>(arg.data_))));
03852         #else
03853             int abs = arg.data_ & 0x7FFF, exp;
03854             if(!abs)
03855                 return arg;
03856             if(abs >= 0x7C00)
03857                 return half(detail::binary, (abs > 0x7C00) ? detail::signal(arg.data_) :
(arg.data_-0x4000));
03858             if(abs >= 0x4500)
03859                 return half(detail::binary,
detail::rounded<half::round_style,true>((arg.data_&0x8000)|0x3BFF, 1, 1));
03860             if(abs < 0x2700)
03861                 return half(detail::binary, detail::rounded<half::round_style,true>(arg.data_-1, 1, 1));
03862             if(half::round_style != std::round_to_nearest && abs == 0x2D3F)
03863                 return half(detail::binary, detail::rounded<half::round_style,true>(arg.data_-3, 0, 1));
03864             std::pair<detail::uint32,detail::uint32> mm = detail::hyperbolic_args(abs, exp, 27);
03865             detail::uint32 my = mm.first - mm.second - (half::round_style!=std::round_to_nearest), mx =
mm.first + mm.second, i = (~mx&0xFFFFFFFF) >> 31;
03866             for(exp+=13; my<0x80000000; my<=1,--exp) ;
03867             mx = (mx>>i) | 0x80000000;
03868             return half(detail::binary, detail::tangent_post<half::round_style>(my, mx, exp-i,
arg.data_&0x8000));
03869         #endif
03870     }
03871
03880     inline half asinh(half arg)
03881     {
03882         #if defined(HALF_ARITHMETIC_TYPE) && HALF_ENABLE_CPP11_CMATH
03883             return half(detail::binary,
detail::float2half<half::round_style>(std::asinh(detail::half2float<detail::internal_t>(arg.data_))));
03884         #else
03885             int abs = arg.data_ & 0x7FFF;
03886             if(!abs || abs >= 0x7C00)
03887                 return (abs > 0x7C00) ? half(detail::binary, detail::signal(arg.data_)) : arg;
03888             if(abs <= 0x2900)
03889                 return half(detail::binary, detail::rounded<half::round_style,true>(arg.data_-1, 1, 1));

```

```

03890         if(half::round_style != std::round_to_nearest)
03891             switch(abs)
03892             {
03893                 case 0x32D4: return half(detail::binary,
03894 detail::rounded<half::round_style,true>(arg.data_-13, 1, 1));
03895                 case 0x3B5B: return half(detail::binary,
03896 detail::rounded<half::round_style,true>(arg.data_-197, 1, 1));
03897             }
03898         return half(detail::binary, detail::area<half::round_style,true>(arg.data_));
03899     #endif
03900 }
03901
03902 inline half acosh(half arg)
03903 {
03904     #if defined(HALF_ARITHMETIC_TYPE) && HALF_ENABLE_CPP11_CMATH
03905         return half(detail::binary,
03906 detail::float2half<half::round_style>(std::acosh(detail::half2float<detail::internal_t>(arg.data_))));
03907     #else
03908         int abs = arg.data_ & 0x7FFF;
03909         if((arg.data_&0x8000) || abs < 0x3C00)
03910             return half(detail::binary, (abs<=0x7C00) ? detail::invalid() :
03911 detail::signal(arg.data_));
03912         if(abs == 0x3C00)
03913             return half(detail::binary, 0);
03914         if(arg.data_ >= 0x7C00)
03915             return (abs>0x7C00) ? half(detail::binary, detail::signal(arg.data_)) : arg;
03916         return half(detail::binary, detail::area<half::round_style,false>(arg.data_));
03917     #endif
03918 }
03919
03920 inline half atanh(half arg)
03921 {
03922     #if defined(HALF_ARITHMETIC_TYPE) && HALF_ENABLE_CPP11_CMATH
03923         return half(detail::binary,
03924 detail::float2half<half::round_style>(std::atanh(detail::half2float<detail::internal_t>(arg.data_))));
03925     #else
03926         int abs = arg.data_ & 0x7FFF, exp = 0;
03927         if(!abs)
03928             return arg;
03929         if(abs >= 0x3C00)
03930             return half(detail::binary, (abs==0x3C00) ? detail::pole(arg.data_&0x8000) : (abs<=0x7C00)
03931 ? detail::invalid() : detail::signal(arg.data_));
03932         if(abs < 0x2700)
03933             return half(detail::binary, detail::rounded<half::round_style,true>(arg.data_, 0, 1));
03934         detail::uint32 m = static_cast<detail::uint32>((abs&0x3FF) | ((abs>0x3FF)<10)) <
03935 ((abs>10)+(abs<=0x3FF)+6), my = 0x80000000 + m, mx = 0x80000000 - m;
03936         for(; mx<0x80000000; mx<=1,++exp);
03937         int i = my >= mx, s;
03938         return half(detail::binary, detail::log2_post<half::round_style,0xB8AA3B2A>(detail::log2(
03939 (detail::divide64(my>i, mx, s)+1)>1, 27)+0x10, exp+i-1, 16, arg.data_&0x8000));
03940     #endif
03941 }
03942
03943 inline half erf(half arg)
03944 {
03945     #if defined(HALF_ARITHMETIC_TYPE) && HALF_ENABLE_CPP11_CMATH
03946         return half(detail::binary,
03947 detail::float2half<half::round_style>(std::erf(detail::half2float<detail::internal_t>(arg.data_))));
03948     #else
03949         unsigned int abs = arg.data_ & 0x7FFF;
03950         if(!abs || abs >= 0x7C00)
03951             return (abs>=0x7C00) ? half(detail::binary, (abs==0x7C00) ? (arg.data_-0x4000) :
03952 detail::signal(arg.data_)) : arg;
03953         if(abs >= 0x4200)
03954             return half(detail::binary,
03955 detail::rounded<half::round_style,true>((arg.data_&0x8000)|0x3BFF, 1, 1));
03956         return half(detail::binary, detail::erf<half::round_style,false>(arg.data_));
03957     #endif
03958 }
03959
03960 inline half erfc(half arg)
03961 {
03962     #if defined(HALF_ARITHMETIC_TYPE) && HALF_ENABLE_CPP11_CMATH
03963         return half(detail::binary,
03964 detail::float2half<half::round_style>(std::erfc(detail::half2float<detail::internal_t>(arg.data_))));
03965     #else
03966         unsigned int abs = arg.data_ & 0x7FFF, sign = arg.data_ & 0x8000;
03967         if(abs >= 0x7C00)
03968             return (abs>=0x7C00) ? half(detail::binary, (abs==0x7C00) ? (sign>1) :
03969 detail::signal(arg.data_)) : arg;
03970         if(!abs)
03971             return half(detail::binary, 0x3C00);
03972         if(abs >= 0x4400)
03973             return half(detail::binary, detail::rounded<half::round_style,true>((sign>1)-(sign>15),
03974 sign>15, 1));
03975         return half(detail::binary, detail::erf<half::round_style,true>(arg.data_));
04000

```

```

04001     #endif
04002     }
04003
04013     inline half lgamma(half arg)
04014     {
04015         #if defined(HALF_ARITHMETIC_TYPE) && HALF_ENABLE_CPP11_CMATH
04016             return half(detail::binary,
04017 detail::float2half<half::round_style>(std::lgamma(detail::half2float<detail::internal_t>(arg.data_))));
04018         #else
04019             int abs = arg.data_ & 0x7FFF;
04020             if(abs >= 0x7C00)
04021                 return half(detail::binary, (abs==0x7C00) ? 0x7C00 : detail::signal(arg.data_));
04022             if(!abs || arg.data_ >= 0xE400 || (arg.data_ >= 0xBC00 && !(abs & ((1<<(25-(abs>>10)))-1))))
04023                 return half(detail::binary, detail::pole());
04024             if(arg.data_ == 0x3C00 || arg.data_ == 0x4000)
04025                 return half(detail::binary, 0);
04026             return half(detail::binary, detail::gamma<half::round_style,true>(arg.data_));
04027         #endif
04028
04038     inline half tgamma(half arg)
04039     {
04040         #if defined(HALF_ARITHMETIC_TYPE) && HALF_ENABLE_CPP11_CMATH
04041             return half(detail::binary,
04042 detail::float2half<half::round_style>(std::tgamma(detail::half2float<detail::internal_t>(arg.data_))));
04043         #else
04044             unsigned int abs = arg.data_ & 0x7FFF;
04045             if(!abs)
04046                 return half(detail::binary, detail::pole(arg.data_));
04047             if(abs >= 0x7C00)
04048                 return (arg.data_==0x7C00) ? arg : half(detail::binary, detail::signal(arg.data_));
04049             if(arg.data_ >= 0xE400 || (arg.data_ >= 0xBC00 && !(abs & ((1<<(25-(abs>>10)))-1))))
04050                 return half(detail::binary, detail::invalid());
04051             if(arg.data_ >= 0xCA80)
04052                 return half(detail::binary,
04053 detail::underflow<half::round_style>((1-((abs>>(25-(abs>>10)))&1))<<15));
04054             if(arg.data_ <= 0x100 || (arg.data_ >= 0x4900 && arg.data_ < 0x8000))
04055                 return half(detail::binary, detail::overflow<half::round_style>());
04056             if(arg.data_ == 0x3C00)
04057                 return arg;
04058             return half(detail::binary, detail::gamma<half::round_style,false>(arg.data_));
04059         #endif
04060     }
04061
04071     inline half ceil(half arg) { return half(detail::binary,
04072 detail::integral<std::round_toward_infinity,true,true>(arg.data_)); }
04073
04079     inline half floor(half arg) { return half(detail::binary,
04080 detail::integral<std::round_toward_neg_infinity,true,true>(arg.data_)); }
04081
04087     inline half trunc(half arg) { return half(detail::binary,
04088 detail::integral<std::round_toward_zero,true,true>(arg.data_)); }
04089
04095     inline half round(half arg) { return half(detail::binary,
04096 detail::integral<std::round_to_nearest,false,true>(arg.data_)); }
04097
04102     inline long lround(half arg) { return
04103 detail::half2int<std::round_to_nearest,false,false,long>(arg.data_); }
04104
04110     inline half rint(half arg) { return half(detail::binary,
04111 detail::integral<half::round_style,true,true>(arg.data_)); }
04112
04118     inline long lrint(half arg) { return
04119 detail::half2int<half::round_style,true,true,long>(arg.data_); }
04120
04125     inline half nearbyint(half arg) { return half(detail::binary,
04126 detail::integral<half::round_style,true,false>(arg.data_)); }
04127
04132     #if HALF_ENABLE_CPP11_LONG_LONG
04133     inline long long llround(half arg) { return
04134 detail::half2int<std::round_to_nearest,false,false,long long>(arg.data_); }
04135
04140     inline long long llrint(half arg) { return detail::half2int<half::round_style,true,true,long
04141 long>(arg.data_); }
04142     #endif
04143
04154     inline half frexp(half arg, int *exp)
04155     {
04156         *exp = 0;
04157         unsigned int abs = arg.data_ & 0x7FFF;
04158         if(abs >= 0x7C00 || !abs)
04159             return (abs>0x7C00) ? half(detail::binary, detail::signal(arg.data_)) : arg;
04160         for(; abs<0x400; abs<=1,--*exp) ;
04161         *exp += (abs>>10) - 14;
04162         return half(detail::binary, (arg.data_&0x8000)|0x3800|(abs&0x3FF));
04163     }

```

```

04164
04174 inline half scalbln(half arg, long exp)
04175 {
04176     unsigned int abs = arg.data_ & 0x7FFF, sign = arg.data_ & 0x8000;
04177     if(abs >= 0x7C00 || !abs)
04178         return (abs>0x7C00) ? half(detail::binary, detail::signal(arg.data_)) : arg;
04179     for(; abs<0x400; abs<=1,--exp) ;
04180     exp += abs > 10;
04181     if(exp > 30)
04182         return half(detail::binary, detail::overflow<half::round_style>(sign));
04183     else if(exp < -10)
04184         return half(detail::binary, detail::underflow<half::round_style>(sign));
04185     else if(exp > 0)
04186         return half(detail::binary, sign | (exp<10) | (abs<0x3FF));
04187     unsigned int m = (abs<0x3FF) | 0x400;
04188     return half(detail::binary, detail::rounded<half::round_style,false>(sign | (m>>(1-exp)),
04189 (m>>-exp)&1, (m&((1<-exp)-1))!=0));
04189 }
04190
04200 inline half scalbn(half arg, int exp) { return scalbln(arg, exp); }
04201
04211 inline half ldexp(half arg, int exp) { return scalbln(arg, exp); }
04212
04219 inline half modf(half arg, half *iptr)
04220 {
04221     unsigned int abs = arg.data_ & 0x7FFF;
04222     if(abs > 0x7C00)
04223     {
04224         arg = half(detail::binary, detail::signal(arg.data_));
04225         return *iptr = arg, arg;
04226     }
04227     if(abs >= 0x6400)
04228         return *iptr = arg, half(detail::binary, arg.data_&0x8000);
04229     if(abs < 0x3C00)
04230         return iptr->data_ = arg.data_ & 0x8000, arg;
04231     unsigned int exp = abs > 10, mask = (1<(25-exp)) - 1, m = arg.data_ & mask;
04232     iptr->data_ = arg.data_ & ~mask;
04233     if(!m)
04234         return half(detail::binary, arg.data_&0x8000);
04235     for(; m<0x400; m<=1,--exp) ;
04236     return half(detail::binary, (arg.data_&0x8000) | (exp<10) | (m<0x3FF));
04237 }
04238
04247 inline int ilogb(half arg)
04248 {
04249     int abs = arg.data_ & 0x7FFF, exp;
04250     if(!abs || abs >= 0x7C00)
04251     {
04252         detail::raise(FE_INVALID);
04253         return !abs ? FP_ILOGB0 : (abs==0x7C00) ? INT_MAX : FP_ILOGBNAN;
04254     }
04255     for(exp=(abs>10)-15; abs<0x200; abs<=1,--exp) ;
04256     return exp;
04257 }
04258
04265 inline half logb(half arg)
04266 {
04267     int abs = arg.data_ & 0x7FFF, exp;
04268     if(!abs)
04269         return half(detail::binary, detail::pole(0x8000));
04270     if(abs >= 0x7C00)
04271         return half(detail::binary, (abs==0x7C00) ? 0x7C00 : detail::signal(arg.data_));
04272     for(exp=(abs>10)-15; abs<0x200; abs<=1,--exp) ;
04273     unsigned int value = static_cast<unsigned>(exp<0) << 15;
04274     if(exp)
04275     {
04276         unsigned int m = std::abs(exp) << 6;
04277         for(exp=18; m<0x400; m<=1,--exp) ;
04278         value |= (exp<10) + m;
04279     }
04280     return half(detail::binary, value);
04281 }
04282
04291 inline half nextafter(half from, half to)
04292 {
04293     int fabs = from.data_ & 0x7FFF, tabs = to.data_ & 0x7FFF;
04294     if(fabs > 0x7C00 || tabs > 0x7C00)
04295         return half(detail::binary, detail::signal(from.data_, to.data_));
04296     if(from.data_ == to.data_ || !(fabs|tabs))
04297         return to;
04298     if(!fabs)
04299     {
04300         detail::raise(FE_UNDERFLOW, !HALF_ERRHANDLING_UNDERFLOW_TO_INEXACT);
04301         return half(detail::binary, (to.data_&0x8000)+1);
04302     }
04303     unsigned int out = from.data_ + (((from.data_>15)^static_cast<unsigned>(
04304

```



```

    (from.data_^(0x8000|(0x8000-(from.data_»15))))<(to.data_^(0x8000|(0x8000-(to.data_»15))))><1> - 1;
04305     detail::raise(FE_OVERFLOW, fabs<0x7C00 && (out&0x7C00)==0x7C00);
04306     detail::raise(FE_UNDERFLOW, !HALF_ERRHANDLING_UNDERFLOW_TO_INEXACT && (out&0x7C00)<0x400);
04307     return half(detail::binary, out);
04308 }
04309
04310 inline half nexttoward(half from, long double to)
04311 {
04320     int fabs = from.data_ & 0x7FFF;
04321     if(fabs > 0x7C00)
04322         return half(detail::binary, detail::signal(from.data_));
04323     long double lfrom = static_cast<long double>(from);
04324     if(detail::builtin_isnan(to) || lfrom == to)
04325         return half(static_cast<float>(to));
04326     if(!fabs)
04327     {
04328         detail::raise(FE_UNDERFLOW, !HALF_ERRHANDLING_UNDERFLOW_TO_INEXACT);
04329         return half(detail::binary, (static_cast<unsigned>(detail::builtin_signbit(to))<15)+1);
04330     }
04331     unsigned int out = from.data_ + (((from.data_»15)^static_cast<unsigned>(lfrom<to))<1) - 1;
04332     detail::raise(FE_OVERFLOW, (out&0x7FFF)==0x7C00);
04333     detail::raise(FE_UNDERFLOW, !HALF_ERRHANDLING_UNDERFLOW_TO_INEXACT && (out&0x7FFF)<0x400);
04334     return half(detail::binary, out);
04335 }
04336
04342 inline HALF_CONSTEXPR half copysign(half x, half y) { return half(detail::binary,
x.data_^(x.data_»y.data_&0x8000)); }
04343
04344
04357 inline HALF_CONSTEXPR int fpclassify(half arg)
04358 {
04359     return !(arg.data_&0x7FFF) ? FP_ZERO :
04360            ((arg.data_&0x7FFF)<0x400) ? FP_SUBNORMAL :
04361            ((arg.data_&0x7FFF)<0x7C00) ? FP_NORMAL :
04362            ((arg.data_&0x7FFF)==0x7C00) ? FP_INFINITE :
04363            FP_NAN;
04364 }
04365
04371 inline HALF_CONSTEXPR bool isfinite(half arg) { return (arg.data_&0x7C00) != 0x7C00; }
04372
04378 inline HALF_CONSTEXPR bool isinf(half arg) { return (arg.data_&0x7FFF) == 0x7C00; }
04379
04385 inline HALF_CONSTEXPR bool isnan(half arg) { return (arg.data_&0x7FFF) > 0x7C00; }
04386
04392 inline HALF_CONSTEXPR bool isnormal(half arg) { return ((arg.data_&0x7C00)!=0) &
((arg.data_&0x7C00)!=0x7C00); }
04393
04399 inline HALF_CONSTEXPR bool signbit(half arg) { return (arg.data_&0x8000) != 0; }
04400
04405
04412 inline HALF_CONSTEXPR bool isgreater(half x, half y)
04413 {
04414     return ((x.data_^(0x8000|(0x8000-(x.data_»15))))+(x.data_»15)) >
((y.data_^(0x8000|(0x8000-(y.data_»15))))+(y.data_»15)) && !isnan(x) && !isnan(y);
04415 }
04416
04423 inline HALF_CONSTEXPR bool isgreaterequal(half x, half y)
04424 {
04425     return ((x.data_^(0x8000|(0x8000-(x.data_»15))))+(x.data_»15)) >=
((y.data_^(0x8000|(0x8000-(y.data_»15))))+(y.data_»15)) && !isnan(x) && !isnan(y);
04426 }
04427
04434 inline HALF_CONSTEXPR bool isless(half x, half y)
04435 {
04436     return ((x.data_^(0x8000|(0x8000-(x.data_»15))))+(x.data_»15)) <
((y.data_^(0x8000|(0x8000-(y.data_»15))))+(y.data_»15)) && !isnan(x) && !isnan(y);
04437 }
04438
04445 inline HALF_CONSTEXPR bool islessequal(half x, half y)
04446 {
04447     return ((x.data_^(0x8000|(0x8000-(x.data_»15))))+(x.data_»15)) <=
((y.data_^(0x8000|(0x8000-(y.data_»15))))+(y.data_»15)) && !isnan(x) && !isnan(y);
04448 }
04449
04456 inline HALF_CONSTEXPR bool islessgreater(half x, half y)
04457 {
04458     return x.data_!=y.data_ && ((x.data_|y.data_)&0x7FFF) && !isnan(x) && !isnan(y);
04459 }
04460
04467 inline HALF_CONSTEXPR bool isunordered(half x, half y) { return isnan(x) || isnan(y); }
04468
04487 template<typename T,typename U> T half_cast(U arg) { return detail::half_caster<T,U>::cast(arg); }
04488
04503 template<typename T,std::float_round_style R,typename U> T half_cast(U arg) { return
detail::half_caster<T,U,R>::cast(arg); }
04505

```

```

04510
04518     inline int feclearexcept(int excepts) { detail::errflags() &= ~excepts; return 0; }
04519
04527     inline int fetestexcept(int excepts) { return detail::errflags() & excepts; }
04528
04538     inline int feraisexexcept(int excepts) { detail::errflags() |= excepts; detail::raise(except);
return 0; }
04539
04548     inline int fegetexceptflag(int *flagp, int excepts) { *flagp = detail::errflags() & excepts;
return 0; }
04549
04559     inline int fesetexceptflag(const int *flagp, int excepts) { detail::errflags() =
(detail::errflags() | (*flagp & excepts)) & (*flagp | ~excepts); return 0; }
04560
04572     inline void fethrowexcept(int excepts, const char *msg = "")
04573     {
04574         excepts &= detail::errflags();
04575         if(excepts & (FE_INVALID|FE_DIVBYZERO))
04576             throw std::domain_error(msg);
04577         if(excepts & FE_OVERFLOW)
04578             throw std::overflow_error(msg);
04579         if(excepts & FE_UNDERFLOW)
04580             throw std::underflow_error(msg);
04581         if(excepts & FE_INEXACT)
04582             throw std::range_error(msg);
04583     }
04585 }
04586
04587
04588 #undef HALF_UNUSED_NOERR
04589 #undef HALF_CONSTEXPR
04590 #undef HALF_CONSTEXPR_CONST
04591 #undef HALF_CONSTEXPR_NOERR
04592 #undef HALF_NOEXCEPT
04593 #undef HALF_NOTHROW
04594 #undef HALF_THREAD_LOCAL
04595 #undef HALF_TWOS_COMPLEMENT_INT
04596 #ifdef HALF_POP_WARNINGS
04597     #pragma warning(pop)
04598     #undef HALF_POP_WARNINGS
04599 #endif
04600
04601 #endif

```

7.6 Light.hpp

```

00001 //
00002 //  NewLight.hpp
00003 //  GL_Scene
00004 //
00005 //  Created by Alonso García on 13/1/25.
00006 //
00007
00008 #pragma once
00009
00010 #include "glm.hpp"
00011 #include <iostream>
00012
00013 #include "Cube.hpp"
00014
00015 namespace udit
00016 {
00026     class Light {
00027     private:
00031         glm::vec3 position;
00032
00036         glm::vec3 color;
00037
00043         float ambientIntensity;
00044
00051         float diffuseIntensity;
00052
00053     public:
00065         Light(const glm::vec3& pos, const glm::vec3& col, float ambient, float diffuse);
00066
00080         static std::shared_ptr <Light> make_light(const glm::vec3& pos, const glm::vec3& col, float
ambient, float diffuse);
00081
00091         void send_to_shader(GLuint program_id) const;
00092     };
00093 }
00094
00095

```

7.7 Mesh.hpp

```

00001 //
00002 // Mesh.hpp
00003 // GL_Geometry
00004 //
00005 // Created by Alonso García on 11/12/24.
00006 //
00007
00008 #pragma once
00009
00010 #include <vector>
00011
00012 #include "glm.hpp"
00013 #include <gtc/matrix_transform.hpp>
00014 #include <gtc/type_ptr.hpp>
00015 #include "glad.h"
00016
00017 #include "Shader.hpp"
00018
00019 namespace udit
00020 {
00021     enum class MeshType
00022     {
00023         BASIC,
00024         MESH,
00025         TERRAIN,
00026         SKYBOX
00027     };
00028
00029     class Mesh
00030     {
00031     private:
00032         enum
00033         {
00034             COORDINATES_VBO,
00035             COLORS_VBO,
00036             NORMALS_VBO,
00037             INDEXES_VBO,
00038             TEXTURE_UV_VBO,
00039             VBO_COUNT
00040         };
00041
00042         MeshType m_mesh_type;
00043
00044     protected:
00045         std::vector<glm::vec3> coordinates;
00046         std::vector<glm::vec3> colors;
00047         std::vector<glm::vec3> normals;
00048         std::vector<GLuint> indices;
00049         std::vector<glm::vec2> texture_uvs;
00050
00051         GLsizei number_of_vertices;
00052
00053         void create_mesh(std::string mesh_name = "");
00054
00055     private:
00056         GLuint vbo_ids[VBO_COUNT];
00057         GLuint vao_id;
00058
00059         glm::mat4 model_view_matrix;
00060         glm::mat4 normal_matrix;
00061
00062         std::shared_ptr < udit::Shader > m_shader;
00063
00064     public:
00065         Mesh();
00066
00067         Mesh(std::string & path);
00068
00069         static std::shared_ptr <Mesh> make_mesh(MeshType type, const std::string &path = "");
00070
00071         virtual ~Mesh();
00072
00073         virtual void translate(glm::vec3 translation);
00074
00075         virtual void rotate(glm::vec3 rotation, float angle);
00076
00077         virtual void scale(glm::vec3 scale);
00078
00079         virtual void update();
00080
00081         virtual void render(glm::mat4 view_matrix);
00082
00083         virtual void resize(glm::mat4 projection_matrix);
00084
00085     };
00086
00087 }

```

```

00195         virtual void set_shader(std::shared_ptr < udit::Shader > shader);
00196
00202         GLuint get_shader_program_id() const;
00203
00211         std::vector < GLint > get_shader_matrix_ids();
00212
00218         glm::mat4 get_model_view_matrix() const { return model_view_matrix; }
00219
00225         void set_model_view_matrix(glm::mat4 matrix) { model_view_matrix = matrix; }
00226
00232         void set_mesh_type(MeshType type) { m_mesh_type = type; }
00233     };
00234
00235 }

```

7.8 Plane.hpp

```

00001 //
00002 // Plane.hpp
00003 // GL_Geometry
00004 //
00005 // Created by Alonso García on 11/12/24.
00006 //
00007
00008 #pragma once
00009
00010 #include "glad.h"
00011
00012 #include "Mesh.hpp"
00013
00014 namespace udit
00015 {
00025     class Plane : public Mesh
00026     {
00027     private:
00031         float width;
00032
00036         float height;
00037
00041         unsigned columns;
00042
00046         unsigned rows;
00047
00048     public:
00054         Plane();
00055
00063         Plane(float size);
00064
00075         Plane(float width, float height, unsigned columns, unsigned rows);
00076
00077     private:
00084         void create_plane();
00085     };
00086
00087 }

```

7.9 GL_Scene/Scene.hpp File Reference

Clase que representa una escena 3D, gestionando objetos como el fondo, terreno, luz, etc.

```

#include <string>
#include "Shader.hpp"
#include "Light.hpp"
#include "Skybox.hpp"
#include "Plane.hpp"

```

Classes

- class [udit::Scene](#)

Representa una escena 3D con un skybox, terreno, luz y otros elementos.

7.9.1 Detailed Description

Clase que representa una escena 3D, gestionando objetos como el fondo, terreno, luz, etc.

Esta clase es responsable de mantener y gestionar la escena 3D, incluyendo el fondo (skybox), el terreno, las luces y el resto de elementos gráficos. Permite actualizar, renderizar y redimensionar la escena, además de configurar las matrices de vista y proyección, y la luz del entorno.

7.10 Scene.hpp

[Go to the documentation of this file.](#)

```

00001 //
00002 //  Scene.hpp
00003 //  GL_Geometry
00004 //
00005 //  Created by Alonso García on 9/12/24.
00006 //
00015
00016 #pragma once
00017
00018 #include <string>
00019 #include "Shader.hpp"
00020 #include "Light.hpp"
00021 #include "Skybox.hpp"
00022 #include "Plane.hpp"
00023
00024 namespace udit
00025 {
00034     class Scene
00035     {
00036     private:
00043         std::vector<std::string> skybox_faces =
00044         {
00045             "skybox_east.jpg", "skybox_west.jpg", "skybox_up.jpg",
00046             "skybox_down.jpg", "skybox_north.jpg", "skybox_south.jpg"
00047         };
00048
00050         float angle = 0.0f;
00051
00053         std::shared_ptr<Skybox> skybox;
00054
00056         std::shared_ptr<Plane> terrain;
00057
00059         std::shared_ptr<Plane> floor;
00060
00062         std::shared_ptr<Mesh> bull;
00063
00065         std::shared_ptr<Light> light;
00066
00068         unsigned width, height;
00069
00071         glm::mat4 view_matrix;
00072
00074         glm::mat4 projection_matrix;
00075
00076     public:
00084         Scene(unsigned width, unsigned height);
00085
00092         void update();
00093
00100         void render();
00101
00109         void resize(unsigned width, unsigned height);
00110
00117         void set_view_matrix(const glm::mat4& view);
00118
00125         void set_projection_matrix(const glm::mat4& projection);
00126
00134         void set_lights(GLuint shader_program_id);
00135     };
00136 }

```

7.11 GL_Scene/Shader.hpp File Reference

Clase que representa un shader en OpenGL, gestionando la compilación y uso de programas de sombreado.

```
#include <iostream>
#include "glad.h"
#include "Texture.hpp"
```

Classes

- class [udit::Shader](#)
Representa un shader program en OpenGL.

Enumerations

- enum class [udit::ShaderType](#) {
 [SKYBOX](#) , [GEOMETRY](#) , [SINGLE_TEXTURE](#) , [TERRAIN](#) ,
 [DEFAULT](#) }
Enumeración que define los diferentes tipos de shaders.

7.11.1 Detailed Description

Clase que representa un shader en OpenGL, gestionando la compilación y uso de programas de sombreado.

La clase [Shader](#) gestiona la creación, compilación y uso de shaders en OpenGL, incluyendo tanto el vertex shader como el fragment shader. Además, permite la gestión de texturas asociadas al shader y la configuración de matrices para la proyección, vista y normales en el contexto de la cámara.

7.11.2 Enumeration Type Documentation

7.11.2.1 ShaderType

```
enum class udit::ShaderType [strong]
```

Enumeración que define los diferentes tipos de shaders.

Define los tipos de shaders que la clase [Shader](#) puede usar para diferentes efectos visuales, como el skybox, geometría, textura única, terreno y por defecto.

Enumerator

SKYBOX	Shader para el skybox.
GEOMETRY	Shader para la geometría.
SINGLE_TEXTURE	Shader para una textura única.
TERRAIN	Shader para el terreno.
DEFAULT	Shader por defecto.

7.12 Shader.hpp

[Go to the documentation of this file.](#)

```

00001 //
00002 //  Shader.hpp
00003 //  GL_Geometry
00004 //
00005 //  Created by Alonso García on 11/12/24.
00006 //
00015
00016 #pragma once
00017
00018 #include <iostream>
00019 #include "glad.h"
00020 #include "Texture.hpp"
00021
00022 namespace udit
00023 {
00031     enum class ShaderType
00032     {
00033         SKYBOX,
00034         GEOMETRY,
00035         SINGLE_TEXTURE,
00036         TERRAIN,
00037         DEFAULT
00038     };
00039
00049     class Shader
00050     {
00051     private:
00053         GLuint program_id;
00054
00056         ShaderType m_type;
00057
00059         std::string m_name;
00060
00062         std::string absolute_path =
00063             "/Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/GL_Scene/";
00065         std::string m_vertex_default_source = absolute_path + "Shader_Default_Vertex.glsl";
00066
00068         std::string m_fragment_default_source = absolute_path + "Shader_Default_Fragment.glsl";
00069
00071         std::string m_vertex_source;
00072
00074         std::string m_fragment_source;
00075
00077         GLint model_view_matrix_id;
00078         GLint projection_matrix_id;
00079         GLint normal_matrix_id;
00080
00082         static const std::string default_vertex_shader_code;
00083         static const std::string default_fragment_shader_code;
00084
00086         std::vector<std::shared_ptr<Texture>> textures;
00087
00088     public:
00095         Shader();
00096
00105         Shader(ShaderType type, const std::string & vertex_source, const std::string &
fragment_source, const std::string & name);
00106
00112         ~Shader();
00113
00125         static std::shared_ptr< Shader > make_shader(
00126             udit::ShaderType type = udit::ShaderType::DEFAULT,
00127             const std::string & vertex_shader = "",
00128             const std::string & fragment_shader = "",
00129             const std::vector<std::string> & texture_paths = {""},
00130             const std::string & name = ""
00131         );
00132
00141         GLuint compile_shaders(const char * vertex_shader_code, const char * fragment_shader_code);
00142
00147         GLint get_model_view_matrix_id() { return model_view_matrix_id; }
00148
00153         GLint get_projection_matrix_id() { return projection_matrix_id; }
00154
00159         GLint get_normal_matrix_id() { return normal_matrix_id; }
00160
00165         GLuint get_program_id() const { return program_id; }
00166
00172         void set_texture(const std::shared_ptr<Texture> & texture);
00173
00179         void use() const;

```

```

00180
00185     void set_texture_scale(float scale);
00186
00191     bool has_textures() { return !textures.empty(); }
00192
00197     void set_name(const std::string & name) { m_name = name; }
00198
00203     std::string get_name() { return m_name; }
00204
00205     private:
00210         void show_compilation_error(GLuint shader_id);
00211
00216         void show_linkage_error(GLuint program_id);
00217     };
00218 }

```

7.13 GL_Scene/Skybox.hpp File Reference

Clase para representar y gestionar un skybox en OpenGL.

```

#include "Cube.hpp"
#include <vector>
#include <string>

```

Classes

- class [udit::Skybox](#)

Representa un skybox, un cubo con texturas aplicadas en sus seis caras.

7.13.1 Detailed Description

Clase para representar y gestionar un skybox en OpenGL.

La clase [Skybox](#) hereda de [Cube](#) y permite la carga y visualización de un cubo que actúa como el fondo de la escena, utilizando una serie de texturas que representan las caras del cielo. Se utiliza para crear una atmósfera inmersiva en la escena renderizada.

7.14 Skybox.hpp

[Go to the documentation of this file.](#)

```

00001 //
00002 //  Skybox.hpp
00003 //  GL_Scene
00004 //
00005 //  Created by Alonso García on 21/12/24.
00006 //
00015
00016 #pragma once
00017
00018 #include "Cube.hpp"
00019 #include <vector>
00020 #include <string>
00021
00022 namespace udit
00023 {
00032     class Skybox : public Cube
00033     {
00034     private:
00036         unsigned int cubemapTexture;
00037

```



```

00039         std::string filepath =
00040             "/Users/alonsooggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/resources/skybox/";
00041     public:
00042         Skybox();
00043
00044         Skybox(float size, const std::vector<std::string>& faces);
00045
00046         unsigned int getCubemapTexture() const { return cubemapTexture; }
00047
00048     private:
00049         void loadCubemap(const std::vector<std::string>& faces);
00050     };
00051 }

```

7.15 GL_Scene/Texture.hpp File Reference

Clase para gestionar las texturas en OpenGL.

```

#include <string>
#include <glad.h>

```

Classes

- class `udit::Texture`
Representa una textura en OpenGL.

Enumerations

- enum class `udit::Texture_Type` { `COLOR`, `HEIGHT` }
Enum que define los tipos de texturas disponibles.

7.15.1 Detailed Description

Clase para gestionar las texturas en OpenGL.

La clase `Texture` permite la carga, enlace y liberación de texturas en OpenGL. Se utiliza para manejar imágenes que se aplican a los objetos 3D en la escena, permitiendo efectos visuales como color, relieve, etc.

7.15.2 Enumeration Type Documentation

7.15.2.1 Texture_Type

```
enum class udit::Texture_Type [strong]
```

Enum que define los tipos de texturas disponibles.

Los tipos de texturas permiten diferenciar entre distintos tipos de efectos visuales:

- `COLOR`: Textura normal, utilizada para representar colores o imágenes en 3D.
- `HEIGHT`: Textura de altura, generalmente utilizada en mapas de relieve.

Enumerator

COLOR	Textura de color (imagen normal).
HEIGHT	Textura de altura (mapa de relieve).

7.16 Texture.hpp

[Go to the documentation of this file.](#)

```

00001 //
00002 // Texture.hpp
00003 // GL_Scene
00004 //
00005 // Created by Alonso García on 24/12/24.
00006 //
00015
00016 #pragma once
00017
00018 #include <string>
00019 #include <glad.h>
00020
00021 namespace udit
00022 {
00031     enum class Texture_Type
00032     {
00033         COLOR,
00034         HEIGHT
00035     };
00036
00045     class Texture
00046     {
00047     private:
00049         bool loaded = false;
00050
00052         Texture_Type m_type;
00053
00054     public:
00065         Texture(const std::string & path, GLenum texture_unit, Texture_Type type =
Texture_Type::COLOR);
00066
00068         ~Texture();
00069
00076         void bind() const;
00077
00084         void unbind() const;
00085
00087         GLuint texture_id;
00088
00090         GLenum texture_unit;
00091
00093         std::string file_path;
00094
00101         void load_texture();
00102
00108         void set_type(Texture_Type type) { m_type = type; }
00109
00115         bool is_loaded() { return loaded; }
00116     };
00117 }
00118

```

7.17 Window.hpp

```

00001 //
00002 // Window.hpp
00003 // GL_Geometry
00004 //
00005 // Created by Alonso García on 9/12/24.
00006 //
00007
00008 #pragma once
00009
00010 #include <SDL.h>

```

```

00011 #include <string>
00012 #include <utility>
00013
00014 namespace udit
00015 {
00016
00017     class Window
00018     {
00019     public:
00020
00021         enum Position
00022         {
00023             UNDEFINED = SDL_WINDOWPOS_UNDEFINED,
00024             CENTERED  = SDL_WINDOWPOS_CENTERED,
00025         };
00026
00027         struct OpenGL_Context_Settings
00028         {
00029             unsigned version_major      = 3;
00030             unsigned version_minor      = 3;
00031             bool      core_profile      = true;
00032             unsigned depth_buffer_size  = 24;
00033             unsigned stencil_buffer_size = 0;
00034             bool      enable_vsync     = true;
00035         };
00036
00037     private:
00038
00039         SDL_Window * window_handle;
00040         SDL_GLContext opengl_context;
00041
00042     public:
00043
00044         Window
00045         (
00046             const std::string & title,
00047             int      left_x,
00048             int      top_y,
00049             unsigned width,
00050             unsigned height,
00051             const OpenGL_Context_Settings & context_details
00052         )
00053         :
00054             Window(title.c_str (), left_x, top_y, width, height, context_details)
00055         {
00056         }
00057
00058         Window
00059         (
00060             const char * title,
00061             int      left_x,
00062             int      top_y,
00063             unsigned width,
00064             unsigned height,
00065             const OpenGL_Context_Settings & context_details
00066         );
00067
00068         ~Window();
00069
00070     public:
00071
00072         Window(const Window & ) = delete;
00073
00074         Window & operator = (const Window & ) = delete;
00075
00076         Window(Window && other) noexcept
00077         {
00078             this->window_handle = std::exchange (other.window_handle, nullptr);
00079             this->opengl_context = std::exchange (other.opengl_context, nullptr);
00080         }
00081
00082         Window & operator = (Window && other) noexcept
00083         {
00084             this->window_handle = std::exchange (other.window_handle, nullptr);
00085             this->opengl_context = std::exchange (other.opengl_context, nullptr);
00086
00087             return * this;
00088         }
00089
00090     public:
00091
00092         void swap_buffers ();
00093
00094     };
00095
00096 }

```


Index

- ~Mesh
 - Mesh, [119](#)
 - udit::Mesh, [125](#)
- ~Shader
 - Shader, [143](#)
 - udit::Shader, [149](#)
- abs
 - half_float, [12](#)
- acos
 - half_float, [13](#)
 - half_float::half, [77](#)
- acosh
 - half_float, [13](#)
 - half_float::half, [77](#)
- angle_arg
 - half.hpp, [174](#)
- area
 - half.hpp, [174](#)
- arithmetic_shift
 - half.hpp, [175](#)
- asin
 - half_float, [14](#)
 - half_float::half, [78](#)
- asinh
 - half_float, [14](#)
 - half_float::half, [78](#)
- atan
 - half_float, [14](#)
 - half_float::half, [78](#)
- atan2
 - half.hpp, [175](#)
 - half_float, [15](#)
 - half_float::half, [79](#)
- atan2_args
 - half.hpp, [176](#)
- atanh
 - half_float, [15](#)
 - half_float::half, [79](#)
- bind
 - Texture, [159](#)
 - udit::Texture, [161](#)
- builtin_isinf
 - half.hpp, [176](#)
- builtin_isnan
 - half.hpp, [176](#)
- builtin_signbit
 - half.hpp, [177](#)
- Camera, [57](#)
 - Camera, [58](#)
 - front, [59](#)
 - get_view_matrix, [58](#)
 - mouse_sensitivity, [59](#)
 - movement_speed, [59](#)
 - pitch, [59](#)
 - position, [59](#)
 - process_keyboard, [58](#)
 - process_mouse_movement, [58](#)
 - right, [60](#)
 - up, [60](#)
 - world_up, [60](#)
 - yaw, [60](#)
 - zoom, [60](#)
- cbrt
 - half_float, [16](#)
 - half_float::half, [80](#)
- ceil
 - half_float, [16](#)
 - half_float::half, [80](#)
- check_underflow
 - half.hpp, [177](#)
- COLOR
 - Texture.hpp, [250](#)
- compile_shaders
 - Shader, [144](#)
 - udit::Shader, [149](#)
- compsignal
 - half.hpp, [178](#)
- copysign
 - half_float, [17](#)
 - half_float::half, [81](#)
- cos
 - half_float, [17](#)
 - half_float::half, [81](#)
- cosh
 - half_float, [17](#)
 - half_float::half, [81](#)
- create_mesh
 - Mesh, [119](#)
 - udit::Mesh, [125](#)
- Cube, [61](#)
 - udit::Cube, [63](#), [64](#)
- DEFAULT
 - Shader.hpp, [246](#)
- divide64
 - half.hpp, [178](#)

- erf
 - half.hpp, 178
 - half_float, 18
 - half_float::half, 82
- erfc
 - half_float, 18
 - half_float::half, 82
- errflags
 - half.hpp, 179
- EventHandler, 65
 - EventHandler, 65
 - handle_events, 65
- exp
 - half_float, 19
 - half_float::half, 83
- exp2
 - half.hpp, 179
 - half_float, 19
 - half_float::half, 83
- exp2_post
 - half.hpp, 179
- expm1
 - half_float, 20
 - half_float::half, 84
- f31
 - half_float::detail::f31, 66, 67
- fabs
 - half_float, 20
 - half_float::half, 84
- fdim
 - half_float, 20
 - half_float::half, 84
- feclearexcept
 - half_float, 21
- fegetexceptflag
 - half_float, 21
- feraiseexcept
 - half_float, 22
- fesetexceptflag
 - half_float, 22
- fetestexcept
 - half_float, 22
- fethrowexcept
 - half_float, 23
- fixed2half
 - half.hpp, 181
- float2half
 - half.hpp, 182
- float2half_impl
 - half.hpp, 182, 183
- floor
 - half_float, 23
 - half_float::half, 85
- fma
 - half_float, 24
 - half_float::half, 85
- fmax
 - half_float, 24
- half_float::half, 86
- fmin
 - half_float, 25
 - half_float::half, 86
- fmod
 - half_float, 25
 - half_float::half, 87
- FP_FAST_FMAH
 - half.hpp, 173
- fpclassify
 - half_float, 25
 - half_float::half, 87
- frexp
 - half_float, 26
 - half_float::half, 88
- front
 - Camera, 59
- gamma
 - half.hpp, 184
- GEOMETRY
 - Shader.hpp, 246
- get_model_view_matrix
 - Mesh, 119
 - udit::Mesh, 125
- get_model_view_matrix_id
 - Shader, 144
 - udit::Shader, 150
- get_name
 - Shader, 144
 - udit::Shader, 150
- get_normal_matrix_id
 - Shader, 144
 - udit::Shader, 150
- get_program_id
 - Shader, 144
 - udit::Shader, 150
- get_projection_matrix_id
 - Shader, 145
 - udit::Shader, 150
- get_shader_matrix_ids
 - Mesh, 120
 - udit::Mesh, 125
- get_shader_program_id
 - Mesh, 120
 - udit::Mesh, 125
- get_view_matrix
 - Camera, 58
- getCubemapTexture
 - Skybox, 155
 - udit::Skybox, 158
- GL_Scene/Camera.hpp, 165
- GL_Scene/Cube.hpp, 166
- GL_Scene/EventHandler.hpp, 166
- GL_Scene/half.hpp, 166, 198
- GL_Scene/Light.hpp, 242
- GL_Scene/Mesh.hpp, 243
- GL_Scene/Plane.hpp, 244
- GL_Scene/Scene.hpp, 244, 245

GL_Scene/Shader.hpp, 246, 247
GL_Scene/Skybox.hpp, 248
GL_Scene/Texture.hpp, 249, 250
GL_Scene/Window.hpp, 250

half

half_float::half, 71

half.hpp

angle_arg, 174
area, 174
arithmetic_shift, 175
atan2, 175
atan2_args, 176
builtin_isinf, 176
builtin_isnan, 176
builtin_signbit, 177
check_underflow, 177
compsignal, 178
divide64, 178
erf, 178
errflags, 179
exp2, 179
exp2_post, 179
fixed2half, 181
float2half, 182
float2half_impl, 182, 183
FP_FAST_FMAH, 173
gamma, 184
half2float, 184
half2float_impl, 186
half2int, 187
HALF_ENABLE_F16C_INTRINSICS, 173
HALF_ERRHANDLING_OVERFLOW_TO_INEXACT, 173
HALF_ERRHANDLING_UNDERFLOW_TO_INEXACT, 173
HALF_ROUND_STYLE, 173
HLF_ROUNDS, 174
HUGE_VALH, 174
hyperbolic_args, 187
hypot_post, 188
int2half, 188
integral, 189
invalid, 189
log2, 190
log2_post, 190
mod, 191
mulhi, 191
multiply64, 191
overflow, 192
pole, 192
raise, 193
rounded, 193
select, 194
sign_mask, 194
signal, 194, 195
sincos, 196
sqrt, 196
tangent_post, 196

underflow, 197

half2float

half.hpp, 184

half2float_impl

half.hpp, 186

half2int

half.hpp, 187

half_cast

half_float, 26, 27

HALF_ENABLE_F16C_INTRINSICS

half.hpp, 173

HALF_ERRHANDLING_OVERFLOW_TO_INEXACT

half.hpp, 173

HALF_ERRHANDLING_UNDERFLOW_TO_INEXACT

half.hpp, 173

half_float, 9

abs, 12
acos, 13
acosh, 13
asin, 14
asinh, 14
atan, 14
atan2, 15
atanh, 15
cbrt, 16
ceil, 16
copysign, 17
cos, 17
cosh, 17
erf, 18
erfc, 18
exp, 19
exp2, 19
expm1, 20
fabs, 20
fdim, 20
feclearexcept, 21
fegetexceptflag, 21
feraiseexcept, 22
fesetexceptflag, 22
fetestexcept, 22
fethrowexcept, 23
floor, 23
fma, 24
fmax, 24
fmin, 25
fmod, 25
fpclassify, 25
frexp, 26
half_cast, 26, 27
hypot, 28
ilogb, 29
isfinite, 29
isgreater, 29
isgreaterequal, 30
isinf, 30
isless, 30
islessequal, 31

- islessgreater, 31
- isnan, 31
- isnormal, 32
- isunordered, 32
- ldexp, 32
- lgamma, 33
- log, 33
- log10, 34
- log1p, 34
- log2, 35
- logb, 35
- lrint, 36
- lround, 36
- modf, 36
- nanh, 37
- nearbyint, 37
- nextafter, 37
- nexttoward, 38
- operator!=, 38
- operator<, 41
- operator<<, 42
- operator<=, 42
- operator>, 43
- operator>>, 44
- operator>=, 43
- operator+, 39
- operator-, 40
- operator/, 41
- operator==, 42
- operator*, 39
- pow, 44
- remainder, 45
- remquo, 45
- rint, 46
- round, 46
- rsqrt, 46
- scalbln, 47
- scalbn, 47
- signbit, 48
- sin, 48
- sincos, 49
- sinh, 49
- sqrt, 50
- tan, 50
- tanh, 50
- tgamma, 51
- trunc, 51
- half_float::detail::binary_t, 53
- half_float::detail::bits< const T >, 54
- half_float::detail::bits< const volatile T >, 54
- half_float::detail::bits< double >, 55
- half_float::detail::bits< float >, 55
- half_float::detail::bits< T >, 53
- half_float::detail::bits< volatile T >, 56
- half_float::detail::bool_type< bool >, 56
- half_float::detail::conditional< bool, T, typename >, 61
- half_float::detail::conditional< false, T, F >, 61
- half_float::detail::f31, 66
- f31, 66, 67
- operator+, 67
- operator-, 67
- operator/, 68
- operator*, 67
- half_float::detail::half_caster< half, half, R >, 112
- half_float::detail::half_caster< half, U, R >, 112
- half_float::detail::half_caster< T, half, R >, 112
- half_float::detail::half_caster< T, U, R >, 111
- half_float::detail::is_float< const T >, 113
- half_float::detail::is_float< const volatile T >, 114
- half_float::detail::is_float< double >, 114
- half_float::detail::is_float< float >, 114
- half_float::detail::is_float< long double >, 115
- half_float::detail::is_float< typename >, 113
- half_float::detail::is_float< volatile T >, 115
- half_float::half, 68
- acos, 77
- acosh, 77
- asin, 78
- asinh, 78
- atan, 78
- atan2, 79
- atanh, 79
- cbrt, 80
- ceil, 80
- copysign, 81
- cos, 81
- cosh, 81
- erf, 82
- erfc, 82
- exp, 83
- exp2, 83
- expm1, 84
- fabs, 84
- fdim, 84
- floor, 85
- fma, 85
- fmax, 86
- fmin, 86
- fmod, 87
- fpclassify, 87
- frexp, 88
- half, 71
- hypot, 88, 89
- ilogb, 89
- isfinite, 90
- isgreater, 90
- isgreaterequal, 90
- isinf, 91
- isless, 91
- islessequal, 91
- islessgreater, 92
- isnan, 92
- isnormal, 92
- lgamma, 93
- log, 93
- log10, 94

- log1p, [94](#)
- log2, [95](#)
- logb, [95](#)
- lrint, [96](#)
- lround, [96](#)
- modf, [96](#)
- nanh, [97](#)
- nearbyint, [97](#)
- nextafter, [97](#)
- nexttoward, [98](#)
- operator float, [71](#)
- operator!=, [98](#)
- operator<, [101](#)
- operator<<, [101](#)
- operator<=, [102](#)
- operator>, [103](#)
- operator>>, [104](#)
- operator>=, [103](#)
- operator+, [99](#)
- operator++, [72](#), [73](#)
- operator+=, [73](#)
- operator-, [100](#)
- operator--, [74](#)
- operator=, [74](#), [75](#)
- operator/, [100](#)
- operator/=, [75](#), [76](#)
- operator=, [76](#)
- operator==, [102](#)
- operator*, [99](#)
- operator*=, [71](#), [72](#)
- pow, [104](#)
- remainder, [105](#)
- remquo, [105](#)
- rint, [106](#)
- round, [106](#)
- rsqrt, [106](#)
- scalbln, [107](#)
- signbit, [107](#)
- sin, [108](#)
- sincos, [108](#)
- sinh, [108](#)
- sqrt, [109](#)
- tan, [109](#)
- tanh, [110](#)
- tgamma, [110](#)
- trunc, [111](#)
- HALF_ROUND_STYLE
 - half.hpp, [173](#)
- handle_events
 - EventHandler, [65](#)
- has_textures
 - Shader, [145](#)
 - udit::Shader, [151](#)
- HEIGHT
 - Texture.hpp, [250](#)
- HLF_ROUNDS
 - half.hpp, [174](#)
- HUGE_VALH
 - half.hpp, [174](#)
- hyperbolic_args
 - half.hpp, [187](#)
- hypot
 - half_float, [28](#)
 - half_float::half, [88](#), [89](#)
- hypot_post
 - half.hpp, [188](#)
- ilogb
 - half_float, [29](#)
 - half_float::half, [89](#)
- int2half
 - half.hpp, [188](#)
- integral
 - half.hpp, [189](#)
- invalid
 - half.hpp, [189](#)
- is_loaded
 - Texture, [159](#)
 - udit::Texture, [161](#)
- isfinite
 - half_float, [29](#)
 - half_float::half, [90](#)
- isgreater
 - half_float, [29](#)
 - half_float::half, [90](#)
- isgreaterequal
 - half_float, [30](#)
 - half_float::half, [90](#)
- isinf
 - half_float, [30](#)
 - half_float::half, [91](#)
- isless
 - half_float, [30](#)
 - half_float::half, [91](#)
- islessequal
 - half_float, [31](#)
 - half_float::half, [91](#)
- islessgreater
 - half_float, [31](#)
 - half_float::half, [92](#)
- isnan
 - half_float, [31](#)
 - half_float::half, [92](#)
- isnormal
 - half_float, [32](#)
 - half_float::half, [92](#)
- isunordered
 - half_float, [32](#)
- ldexp
 - half_float, [32](#)
- lgamma
 - half_float, [33](#)
 - half_float::half, [93](#)
- Light
 - udit::Light, [116](#)
- load_texture

- Texture, 159
- udit::Texture, 162
- log
 - half_float, 33
 - half_float::half, 93
- log10
 - half_float, 34
 - half_float::half, 94
- log1p
 - half_float, 34
 - half_float::half, 94
- log2
 - half.hpp, 190
 - half_float, 35
 - half_float::half, 95
- log2_post
 - half.hpp, 190
- logb
 - half_float, 35
 - half_float::half, 95
- lrint
 - half_float, 36
 - half_float::half, 96
- lround
 - half_float, 36
 - half_float::half, 96
- make_light
 - udit::Light, 116
- make_mesh
 - Mesh, 120
 - udit::Mesh, 126
- make_shader
 - Shader, 145
 - udit::Shader, 151
- Mesh, 117
 - ~Mesh, 119
 - create_mesh, 119
 - get_model_view_matrix, 119
 - get_shader_matrix_ids, 120
 - get_shader_program_id, 120
 - make_mesh, 120
 - Mesh, 119
 - render, 120
 - resize, 121
 - rotate, 121
 - scale, 121
 - set_mesh_type, 121
 - set_model_view_matrix, 122
 - set_shader, 122
 - translate, 122
 - udit::Mesh, 124
 - update, 122
- mod
 - half.hpp, 191
- modf
 - half_float, 36
 - half_float::half, 96
- mouse_sensitivity
 - Camera, 59
- movement_speed
 - Camera, 59
- mulhi
 - half.hpp, 191
- multiply64
 - half.hpp, 191
- nanh
 - half_float, 37
 - half_float::half, 97
- nearbyint
 - half_float, 37
 - half_float::half, 97
- nextafter
 - half_float, 37
 - half_float::half, 97
- nexttoward
 - half_float, 38
 - half_float::half, 98
- operator float
 - half_float::half, 71
- operator!=
 - half_float, 38
 - half_float::half, 98
- operator<
 - half_float, 41
 - half_float::half, 101
- operator<<
 - half_float, 42
 - half_float::half, 101
- operator<=
 - half_float, 42
 - half_float::half, 102
- operator>
 - half_float, 43
 - half_float::half, 103
- operator>>
 - half_float, 44
 - half_float::half, 104
- operator>=
 - half_float, 43
 - half_float::half, 103
- operator+
 - half_float, 39
 - half_float::detail::f31, 67
 - half_float::half, 99
- operator++
 - half_float::half, 72, 73
- operator+=
 - half_float::half, 73
- operator-
 - half_float, 40
 - half_float::detail::f31, 67
 - half_float::half, 100
- operator--
 - half_float::half, 74
- operator-=

- half_float::half, [74](#), [75](#)
- operator/
 - half_float, [41](#)
 - half_float::detail::f31, [68](#)
 - half_float::half, [100](#)
- operator/= [half_float::half](#), [75](#), [76](#)
- operator=
 - half_float::half, [76](#)
- operator==
 - half_float, [42](#)
 - half_float::half, [102](#)
- operator*
 - half_float, [39](#)
 - half_float::detail::f31, [67](#)
 - half_float::half, [99](#)
- operator*=
 - half_float::half, [71](#), [72](#)
- overflow
 - half.hpp, [192](#)
- pitch
 - Camera, [59](#)
- Plane, [131](#)
 - Plane, [133](#)
 - udit::Plane, [136](#)
- pole
 - half.hpp, [192](#)
- position
 - Camera, [59](#)
- pow
 - half_float, [44](#)
 - half_float::half, [104](#)
- process_keyboard
 - Camera, [58](#)
- process_mouse_movement
 - Camera, [58](#)
- raise
 - half.hpp, [193](#)
- remainder
 - half_float, [45](#)
 - half_float::half, [105](#)
- remquo
 - half_float, [45](#)
 - half_float::half, [105](#)
- render
 - Mesh, [120](#)
 - Scene, [138](#)
 - udit::Mesh, [126](#)
 - udit::Scene, [140](#)
- resize
 - Mesh, [121](#)
 - Scene, [138](#)
 - udit::Mesh, [126](#)
 - udit::Scene, [140](#)
- right
 - Camera, [60](#)
- rint
 - half_float, [46](#)
 - half_float::half, [106](#)
- rotate
 - Mesh, [121](#)
 - udit::Mesh, [127](#)
- round
 - half_float, [46](#)
 - half_float::half, [106](#)
- rounded
 - half.hpp, [193](#)
- rsqrt
 - half_float, [46](#)
 - half_float::half, [106](#)
- scalbln
 - half_float, [47](#)
 - half_float::half, [107](#)
- scalbn
 - half_float, [47](#)
- scale
 - Mesh, [121](#)
 - udit::Mesh, [127](#)
- Scene, [137](#)
 - render, [138](#)
 - resize, [138](#)
 - Scene, [137](#)
 - set_lights, [138](#)
 - set_projection_matrix, [138](#)
 - set_view_matrix, [139](#)
 - udit::Scene, [140](#)
 - update, [139](#)
- select
 - half.hpp, [194](#)
- send_to_shader
 - udit::Light, [117](#)
- set_lights
 - Scene, [138](#)
 - udit::Scene, [141](#)
- set_mesh_type
 - Mesh, [121](#)
 - udit::Mesh, [127](#)
- set_model_view_matrix
 - Mesh, [122](#)
 - udit::Mesh, [127](#)
- set_name
 - Shader, [145](#)
 - udit::Shader, [151](#)
- set_projection_matrix
 - Scene, [138](#)
 - udit::Scene, [141](#)
- set_shader
 - Mesh, [122](#)
 - udit::Mesh, [128](#)
- set_texture
 - Shader, [147](#)
 - udit::Shader, [152](#)
- set_texture_scale
 - Shader, [147](#)
 - udit::Shader, [152](#)

- set_type
 - Texture, 160
 - udit::Texture, 162
- set_view_matrix
 - Scene, 139
 - udit::Scene, 141
- Shader, 142
 - ~Shader, 143
 - compile_shaders, 144
 - get_model_view_matrix_id, 144
 - get_name, 144
 - get_normal_matrix_id, 144
 - get_program_id, 144
 - get_projection_matrix_id, 145
 - has_textures, 145
 - make_shader, 145
 - set_name, 145
 - set_texture, 147
 - set_texture_scale, 147
 - Shader, 143
 - udit::Shader, 149
 - use, 147
- Shader.hpp
 - DEFAULT, 246
 - GEOMETRY, 246
 - ShaderType, 246
 - SINGLE_TEXTURE, 246
 - SKYBOX, 246
 - TERRAIN, 246
- ShaderType
 - Shader.hpp, 246
- sign_mask
 - half.hpp, 194
- signal
 - half.hpp, 194, 195
- signbit
 - half_float, 48
 - half_float::half, 107
- sin
 - half_float, 48
 - half_float::half, 108
- sincos
 - half.hpp, 196
 - half_float, 49
 - half_float::half, 108
- SINGLE_TEXTURE
 - Shader.hpp, 246
- sinh
 - half_float, 49
 - half_float::half, 108
- SKYBOX
 - Shader.hpp, 246
- Skybox, 152
 - getCubemapTexture, 155
 - Skybox, 154
 - udit::Skybox, 157
- sqrt
 - half.hpp, 196
- half_float, 50
 - half_float::half, 109
- std, 52
- std::numeric_limits< half_float::half >, 128
- tan
 - half_float, 50
 - half_float::half, 109
- tangent_post
 - half.hpp, 196
- tanh
 - half_float, 50
 - half_float::half, 110
- TERRAIN
 - Shader.hpp, 246
- Texture, 158
 - bind, 159
 - is_loaded, 159
 - load_texture, 159
 - set_type, 160
 - Texture, 159
 - udit::Texture, 161
 - unbind, 160
- Texture.hpp
 - COLOR, 250
 - HEIGHT, 250
 - Texture_Type, 249
- Texture_Type
 - Texture.hpp, 249
- tgamma
 - half_float, 51
 - half_float::half, 110
- translate
 - Mesh, 122
 - udit::Mesh, 128
- trunc
 - half_float, 51
 - half_float::half, 111
- udit::Cube, 62
 - Cube, 63, 64
- udit::Light, 115
 - Light, 116
 - make_light, 116
 - send_to_shader, 117
- udit::Mesh, 123
 - ~Mesh, 125
 - create_mesh, 125
 - get_model_view_matrix, 125
 - get_shader_matrix_ids, 125
 - get_shader_program_id, 125
 - make_mesh, 126
 - Mesh, 124
 - render, 126
 - resize, 126
 - rotate, 127
 - scale, 127
 - set_mesh_type, 127
 - set_model_view_matrix, 127

- set_shader, [128](#)
- translate, [128](#)
- update, [128](#)
- udit::Plane, [134](#)
 - Plane, [136](#)
- udit::Scene, [139](#)
 - render, [140](#)
 - resize, [140](#)
 - Scene, [140](#)
 - set_lights, [141](#)
 - set_projection_matrix, [141](#)
 - set_view_matrix, [141](#)
 - update, [141](#)
- udit::Shader, [147](#)
 - ~Shader, [149](#)
 - compile_shaders, [149](#)
 - get_model_view_matrix_id, [150](#)
 - get_name, [150](#)
 - get_normal_matrix_id, [150](#)
 - get_program_id, [150](#)
 - get_projection_matrix_id, [150](#)
 - has_textures, [151](#)
 - make_shader, [151](#)
 - set_name, [151](#)
 - set_texture, [152](#)
 - set_texture_scale, [152](#)
 - Shader, [149](#)
 - use, [152](#)
- udit::Skybox, [155](#)
 - getCubemapTexture, [158](#)
 - Skybox, [157](#)
- udit::Texture, [160](#)
 - bind, [161](#)
 - is_loaded, [161](#)
 - load_texture, [162](#)
 - set_type, [162](#)
 - Texture, [161](#)
 - unbind, [162](#)
- udit::Window, [162](#)
 - Window, [163](#)
- udit::Window::OpenGL_Context_Settings, [130](#)
- unbind
 - Texture, [160](#)
 - udit::Texture, [162](#)
- underflow
 - half.hpp, [197](#)
- up
 - Camera, [60](#)
- update
 - Mesh, [122](#)
 - Scene, [139](#)
 - udit::Mesh, [128](#)
 - udit::Scene, [141](#)
- use
 - Shader, [147](#)
 - udit::Shader, [152](#)
- Window, [164](#)
 - udit::Window, [163](#)
- Window, [164](#)
- Window::OpenGL_Context_Settings, [131](#)
- world_up
 - Camera, [60](#)
- yaw
 - Camera, [60](#)
- zoom
 - Camera, [60](#)