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()	
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 fmax()
5.1.2.30 fmin()
5.1.2.31 fmod()
5.1.2.32 fpclassify()
5.1.2.33 frexp()
5.1.2.34 half_cast() [1/2]
5.1.2.35 half_cast() [2/2]
5.1.2.36 hypot() [1/2]
5.1.2.37 hypot() [2/2]
5.1.2.38 ilogb()
5.1.2.39 isfinite()
5.1.2.40 isgreater()
5.1.2.41 isgreaterequal()
5.1.2.42 isinf()
5.1.2.43 isless()
5.1.2.44 islessequal()
5.1.2.45 islessgreater()
5.1.2.46 isnan()
5.1.2.47 isnormal()
5.1.2.48 isunordered()
5.1.2.49 ldexp()
5.1.2.50 lgamma()
5.1.2.51 log()
5.1.2.52 log10()
5.1.2.53 log1p()
5.1.2.54 log2()
5.1.2.55 logb()
5.1.2.56 lrint()
5.1.2.57 lround()
5.1.2.58 modf()
5.1.2.59 nanh()
5.1.2.60 nearbyint()
5.1.2.61 nextafter()
5.1.2.62 nexttoward()
5.1.2.63 operator"!=()
5.1.2.64 operator*()
5.1.2.65 operator+() [1/2]
5.1.2.66 operator+() [2/2]
5.1.2.67 operator-() [1/2]
5.1.2.68 operator-() [2/2]
5.1.2.69 operator/()
5.1.2.70 operator<()

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
$\textbf{6.10 half_float::} \textbf{detail::} \textbf{conditional} < \textbf{bool}, \textbf{T}, \textbf{typename} > \textbf{Struct Template Reference} \dots \dots \dots$	61
6.10.1 Detailed Description	61
$\textbf{6.11 half_float::} \textbf{detail::} \textbf{conditional} < \textbf{false}, \textbf{T}, \textbf{F} > \textbf{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
6.16.4.21 fma
6.16.4.22 fmax
6.16.4.23 fmin
6.16.4.24 fmod
6.16.4.25 fpclassify
6.16.4.26 frexp
6.16.4.27 hypot [1/2]
6.16.4.28 hypot [2/2]
6.16.4.29 ilogb
6.16.4.30 isfinite
6.16.4.31 isgreater
6.16.4.32 isgreaterequal
6.16.4.33 isinf
6.16.4.34 isless
6.16.4.35 islessequal
6.16.4.36 islessgreater
6.16.4.37 isnan
6.16.4.38 isnormal
6.16.4.39 lgamma
6.16.4.40 log
6.16.4.41 log10
6.16.4.42 log1p
6.16.4.43 log2
6.16.4.44 logb
6.16.4.45 Irint
6.16.4.46 Iround
6.16.4.47 modf
6.16.4.48 nanh
6.16.4.49 nearbyint
6.16.4.50 nextafter
6.16.4.51 nexttoward
6.16.4.52 operator"!=
6.16.4.53 operator*
6.16.4.54 operator+
6.16.4.55 operator- [1/2]
6.16.4.56 operator- [2/2]
6.16.4.57 operator/
6.16.4.58 operator<
6.16.4.59 operator<<
6.16.4.60 operator<=
6.16.4.61 operator==

6.16.4.62 operator>)3
6.16.4.63 operator>=)3
6.16.4.64 operator>>)4
6.16.4.65 pow)4
6.16.4.66 remainder)5
6.16.4.67 remquo)5
6.16.4.68 rint)6
6.16.4.69 round)6
6.16.4.70 rsqrt)6
6.16.4.71 scalbin)7
6.16.4.72 signbit)7
6.16.4.73 sin)8
6.16.4.74 sincos)8
6.16.4.75 sinh)9
6.16.4.76 sqrt)9
6.16.4.77 tan)9
6.16.4.78 tanh	0
6.16.4.79 tgamma	0
6.16.4.80 trunc	1
$6.17 \ half_float:: detail:: half_caster < T, \ U, \ R > Struct \ Template \ Reference \\ \ \ldots \\ \ \ldots \\ \ \ldots \\ \ 11$	1
6.17.1 Detailed Description	1
$6.18 \ half_float:: detail:: half_caster < half, \ half, \ R > Struct \ Template \ Reference \ . \ . \ . \ . \ . \ . \ . \ . \ . \ $	2
$6.19 \ half_float:: detail:: half_caster < half, \ U, \ R > Struct \ Template \ Reference \ . \ . \ . \ . \ . \ . \ . \ . \ . \ $	2
$6.20 \ half_float:: detail:: half_caster < T, \ half, \ R > Struct \ Template \ Reference \ \dots $	2
$\textbf{6.21 half_float::} \\ \textbf{detail::} \\ \textbf{is_float} \\ < \\ \textbf{typename} \\ > \\ \textbf{Struct Template Reference} \\ \\ \\ \dots \\ \dots \\ \\ \dots \\ \dots \\ \dots \\ \dots \\ \dots \\ \dots $	3
6.21.1 Detailed Description	3
$6.22 \ half_float:: detail:: is_float < const \ T > Struct \ Template \ Reference \ . \ . \ . \ . \ . \ . \ . \ . \ . \ $	3
$6.23 \ half_float:: detail:: is_float < const \ volatile \ T > Struct \ Template \ Reference \ . \ . \ . \ . \ . \ . \ . \ . \ . \ $	4
$\textbf{6.24 half_float::} \\ \textbf{detail::} \\ \textbf{is_float} \\ < \\ \textbf{double} \\ > \\ \textbf{Struct Reference} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\$	4
$\textbf{6.25 half_float::} \\ \textbf{detail::} \\ \textbf{is_float} < \textbf{float} > \\ \textbf{Struct Reference} \\ \dots \\ $	4
$6.26 \ half_float:: detail:: is_float < long \ double > Struct \ Reference \ . \ . \ . \ . \ . \ . \ . \ . \ . \ $	5
$6.27 \ half_float:: detail:: is_float < volatile \ T > Struct \ Template \ Reference \ \dots \ \dots \ \dots \ \dots \ 11$	5
6.28 udit::Light Class Reference	5
6.28.1 Detailed Description	6
6.28.2 Constructor & Destructor Documentation	6
6.28.2.1 Light()	6
6.28.3 Member Function Documentation	6
6.28.3.1 make_light()	6
6.28.3.2 send_to_shader()	7
6.29 Mesh Class Reference	7
6.29.1 Detailed Description	9
6.29.2 Constructor & Destructor Documentation	9

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

6.34.1 Detailed Description	33
6.34.2 Constructor & Destructor Documentation	33
6.34.2.1 Plane() [1/3] 1	33
6.34.2.2 Plane() [2/3] 1	33
6.34.2.3 Plane() [3/3]	33
6.35 udit::Plane Class Reference	34
6.35.1 Detailed Description	36
6.35.2 Constructor & Destructor Documentation	36
6.35.2.1 Plane() [1/3] 1	36
6.35.2.2 Plane() [2/3]	36
6.35.2.3 Plane() [3/3] 1	36
6.36 Scene Class Reference	37
6.36.1 Detailed Description	37
6.36.2 Constructor & Destructor Documentation	37
6.36.2.1 Scene()	37
6.36.3 Member Function Documentation	38
6.36.3.1 render()	38
6.36.3.2 resize()	38
6.36.3.3 set_lights()	38
6.36.3.4 set_projection_matrix()	38
6.36.3.5 set_view_matrix()	39
6.36.3.6 update()	39
6.37 udit::Scene Class Reference	39
6.37.1 Detailed Description	40
6.37.2 Constructor & Destructor Documentation	40
6.37.2.1 Scene()	40
6.37.3 Member Function Documentation	40
6.37.3.1 render()	40
6.37.3.2 resize()	40
6.37.3.3 set_lights()	41
6.37.3.4 set_projection_matrix()	41
6.37.3.5 set_view_matrix()	41
6.37.3.6 update()	42
6.38 Shader Class Reference	42
6.38.1 Detailed Description	43
6.38.2 Constructor & Destructor Documentation	43
6.38.2.1 Shader() [1/2] 1	43
6.38.2.2 Shader() [2/2] 1	43
6.38.2.3 ∼Shader()	43
6.38.3 Member Function Documentation	44
6.38.3.1 compile_shaders()	44
6.38.3.2 get_model_view_matrix_id()	44

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()	8
	6.42 Texture Class Reference	8
	6.42.1 Detailed Description	9
	6.42.2 Constructor & Destructor Documentation	9
	6.42.2.1 Texture()	9
	6.42.3 Member Function Documentation	9
	6.42.3.1 bind()	9
	6.42.3.2 is_loaded()	9
	6.42.3.3 load_texture()	0
	6.42.3.4 set_type()	0
	6.42.3.5 unbind()	0
	6.43 udit::Texture Class Reference	0
	6.43.1 Detailed Description	1
	6.43.2 Constructor & Destructor Documentation	1
	6.43.2.1 Texture()	1
	6.43.3 Member Function Documentation	1
	6.43.3.1 bind()	1
	6.43.3.2 is_loaded()	2
	6.43.3.3 load_texture()	2
	6.43.3.4 set_type()	2
	6.43.3.5 unbind()	2
	6.44 udit::Window Class Reference	2
	6.44.1 Constructor & Destructor Documentation	3
	6.44.1.1 Window()	3
	6.45 Window Class Reference	i 4
	6.45.1 Constructor & Destructor Documentation	4
	6.45.1.1 Window()	4
71	File Documentation 16	5
′	7.1 Camera.hpp	
	7.2 Cube.hpp	
	7.3 EventHandler.hpp	
	7.4 GL Scene/half.hpp File Reference	
	7.4.1 Detailed Description	
	7.4.2 Macro Definition Documentation	
	7.4.2.1 FP FAST FMAH	
	7.4.2.2 HALF_ENABLE_F16C_INTRINSICS	
	7.4.2.3 HALF ERRHANDLING OVERFLOW TO INEXACT	
	7.4.2.4 HALF_ERRHANDLING_UNDERFLOW_TO_INEXACT	
	7.4.2.5 HALF ROUND STYLE	
	7.4.2.6 HLF ROUNDS	
	7.4.2.7 HUGE_VALH	
		•

7.4.3 F	unction 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

Namespace Index

1.1 Namespace List

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

halt_tloa	[
std	
	Extensions to the C++ standard library

2 Namespace Index

Hierarchical Index

2.1 Class Hierarchy

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

half_float::detail::binary_t
$half_float::detail::bits < T > \dots \dots$
half_float::detail::bits < const T >
$half_float::detail::bits < const \ volatile \ T > \dots \dots$
$half_float::detail::bits < volatile \ T > \ \dots \$
half_float::detail::bits < double >
half_float::detail::bool_type< bool >
half_float::detail::bool_type< false >
half_float::detail::is_float< T >
half_float::detail::is_float< const T >
half_float::detail::is_float< const volatile T >
half_float::detail::is_float< volatile T >
half_float::detail::is_float< const T >
half_float::detail::is_float< const volatile T >
half_float::detail::is_float< double >
half_float::detail::is_float < float >
half_float::detail::is_float< long double >
half_float::detail::is_float< volatile T >
half_float::detail::is_float< typename >
half_float::detail::bool_type< true >
half_float::detail::is_float< double >
half_float::detail::is_float < float >
half_float::detail::is_float< long double >
Camera
half_float::detail::conditional< bool, T, typename >
$half_float::detail::conditional < false, T, F > \dots \dots$
half_float::detail::conditional< std::numeric_limits< unsigned int >::digits >
half_float::detail::bits< float >
Cube
EventHandler
half_float::detail::f31
half_float::half
$half_float::detail::half_caster < T, U, R > \dots \dots \dots \dots \dots \dots \dots \dots \dots $
half_float::detail::half_caster< half, half, R >

4 Hierarchical Index

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	_
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
Mindow	164

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
$\label{local-conditional} \textbf{half_float::} detail::conditional < false, T, F > \dots \dots$	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
$half_float:: detail:: half_caster < half, \ half, \ R > \qquad \dots \qquad \dots \qquad \dots \qquad \dots \qquad \dots$	112
$half_float:: detail:: half_caster < half, \ U, \ R > \dots \dots$	112
$half_float::detail::half_caster < T, \ half, \ R > \ \dots \dots$	112
half_float::detail::is_float< typename >	
Type traits for floating-point types	113
	113
$half_float:: detail:: is_float < const \ volatile \ T > \ \dots \ \dots$	114

6 Class Index

half_floa	tt::detail::is_float< double >	114
half_floa	.t::detail::is_float< float >	114
half_floa	.t::detail::is_float< long double >	115
half_floa	ıt::detail::is_float $<$ volatile T $>$	115
udit::Ligh	nt .	
	Clase que representa una fuente de luz en la escena	115
Mesh		
	Clase que representa una malla 3D	117
udit::Mes	sh	
	Clase que representa una malla 3D	123
	-	128
	. – – – –	130
Window:	::OpenGL_Context_Settings	131
Plane		
		131
udit::Pla		
_	Clase que representa un plano 3D	134
Scene		
		137
udit::Sce		
. .	Representa una escena 3D con un skybox, terreno, luz y otros elementos	139
Shader		
1'4 O	-h h - 9 h	142
udit::Sha		4 47
Claubov	Representa un shader program en OpenGL	147
Skybox	Penrocente un alcubey un aube con texturas anlicadas en que cois caras	152
udit::Sky	•	132
uuitSky		155
Texture	riepiesenia un skybox, un cubo con texturas aplicadas en sus seis caras	133
Texture	Representa una textura en OpenGL	158
udit::Tex	·	100
		160
udit::Win	·	162
Window		16/

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 2	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

8 File Index

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 Igamma (half arg)
- half tgamma (half arg)

Rounding

- half ceil (half arg)
- half floor (half arg)
- half trunc (half arg)

- half round (half arg)
- long Iround (half arg)
- half rint (half arg)
- long Irint (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 Idexp (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()

Absolute value. **See also:** Documentation for std::abs.

arg operand

Returns

absolute value of arg

5.1.2.2 acos()

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

See also: Documentation for std::acos.

Parameters

arg function argument

Returns

arc cosine value of arg

Exceptions

FE_INVALID	for signaling NaN or if abs(arg) > 1
FE_OVERFLOW,UNDERFLOW,INEXACT	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

arg function argument

Returns

area cosine value of arg

Exceptions

FE_INVALID	for signaling NaN or arguments <1
FE_OVERFLOW,UNDERFLOW,INEXACT	according to rounding

5.1.2.4 asin()

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

See also: Documentation for std::asin.

Parameters

arg function argument

Returns

arc sine value of arg

Exceptions

FE_INVALID	for signaling NaN or if abs(arg) > 1
FE_OVERFLOW,UNDERFLOW,INEXACT	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

arg function argument

Returns

area sine value of arg

Exceptions

FE_INVALID	for signaling NaN
FE_OVERFLOW,UNDERFLOW,INEXACT	according to rounding

5.1.2.6 atan()

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

See also: Documentation for std::atan.

nt

Returns

arc tangent value of arg

Exceptions

FE_INVALID	for signaling NaN
FE_OVERFLOW,UNDERFLOW,INEXACT	according to rounding

5.1.2.7 atan2()

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

У	numerator
Х	denominator

Returns

arc tangent value

Exceptions

FE_INVALID	if x or y is signaling NaN
FE_OVERFLOW,UNDERFLOW,INEXACT	according to rounding

5.1.2.8 atanh()

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

See also: Documentation for std::atanh.

arg	function argument
-----	-------------------

Returns

area tangent value of arg

Exceptions

FE_INVALID	for signaling NaN or if abs(arg) > 1
FE_DIVBYZERO	for +/-1
FE_OVERFLOW,UNDERFLOW,INEXACT	according to rounding

5.1.2.9 cbrt()

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

See also: Documentation for std::cbrt.

Parameters

arg	function argument
-----	-------------------

Returns

cubic root of arg

Exceptions

FE_INVALID	for signaling NaN
FE_INEXACT	according to rounding

5.1.2.10 ceil()

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

Parameters

arg	half to round

Returns

nearest integer not less than arg

Exceptions

FE_INVALID	for signaling NaN
FE_INEXACT	if value had to be rounded

5.1.2.11 copysign()

Take sign. **See also:** Documentation for std::copysign.

Parameters

X	value to change sign for
У	value to take sign from

Returns

value equal to x in magnitude and to y in sign

5.1.2.12 cos()

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

See also: Documentation for std::cos.

Parameters

```
arg function argument
```

Returns

cosine value of arg

Exceptions

FE_INVALID	for signaling NaN or infinity
FE_OVERFLOW,UNDERFLOW,INEXACT	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.

nt

Returns

hyperbolic cosine value of arg

Exceptions

FE_INVALID	for signaling NaN
FE_OVERFLOW,UNDERFLOW,INEXACT	according to rounding

5.1.2.14 erf()

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 argume	ent
--------------------------------	-----

Returns

error function value of arg

Exceptions

FE_INVALID	for signaling NaN
FE_OVERFLOW,UNDERFLOW,INEXACT	according to rounding

5.1.2.15 erfc()

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.

arg	function argument
-----	-------------------

Returns

1 minus error function value of arg

Exceptions

FE_INVALID	for signaling NaN
FE_OVERFLOW,UNDERFLOW,INEXACT	according to rounding

5.1.2.16 exp()

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

See also: Documentation for std::exp.

Parameters

```
arg function argument
```

Returns

e raised to arg

Exceptions

FE_INVALID	for signaling NaN
FE_OVERFLOW,UNDERFLOW,INEXACT	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

```
arg function argument
```

Returns

2 raised to arg

Exceptions

FE_INVALID	for signaling NaN
FE_OVERFLOW,UNDERFLOW,INEXACT	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

Returns

e raised to arg and subtracted by 1

Exceptions

FE_INVALID	for signaling NaN
FE_OVERFLOW,UNDERFLOW,INEXACT	according to rounding

5.1.2.19 fabs()

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

Parameters

```
arg 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.

X	first operand	
У	second operand	

Returns

x - y or 0 if difference negative

Exceptions

FE↔	according to operator-(half,half)	

5.1.2.21 feclearexcept()

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

excepts	OR of exceptions to clear
---------	---------------------------

Return values

0 all selected flags cleared successfully

5.1.2.22 fegetexceptflag()

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

flagp	adress to store flag state at
excepts	OR of flags to save

Return values

```
0 for success
```

5.1.2.23 feraiseexcept()

Raise exception flags. This raises the specified floating point exceptions and also invokes any additional automatic exception handling as configured with the HALF_ERRHANDLIG_... 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

excepts OR	of exceptions to raise
------------	------------------------

Return values

0 all selected exceptions raised successfully

5.1.2.24 fesetexceptflag()

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

flagp	adress to take flag state from	
excepts	OR of flags to restore	

Return values

```
0 for success
```

5.1.2.25 fetestexcept()

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.

excepts OR of exceptions to test	
----------------------------------	--

Returns

OR of selected exceptions if raised

5.1.2.26 fethrowexcept()

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

excepts	OR of exceptions to test	
msg	error message to use for exception description	

Exceptions

std::domain_error	if FE_INVALID or FE_DIVBYZERO is selected and set
std::overflow_error	if FE_OVERFLOW is selected and set
std::underflow_error	if FE_UNDERFLOW is selected and set
std::range_error	if FE_INEXACT 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

```
arg half to round
```

Returns

nearest integer not greater than arg

FE_INVALID	for signaling NaN
FE_INEXACT	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

Х	first operand	
у	second operand	
Z	third operand	

Returns

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

Exceptions

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

5.1.2.29 fmax()

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

Parameters

X	first operand
У	second operand

Returns

maximum of operands, ignoring quiet NaNs

FE_INVALID	if x or y is signaling NaN
------------	----------------------------

5.1.2.30 fmin()

Minimum of half expressions. **See also:** Documentation for std::fmin.

Parameters

X	first operand
У	second operand

Returns

minimum of operands, ignoring quiet NaNs

Exceptions

5.1.2.31 fmod()

Remainder of division. **See also:** Documentation for std::fmod.

Parameters

Х	first operand
У	second operand

Returns

remainder of floating-point division.

Exceptions

5.1.2.32 fpclassify()

Classify floating-point value. **See also:** Documentation for std::fpclassify.

arg	number to classify
-----	--------------------

Return values

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

5.1.2.33 frexp()

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

Parameters

arg	number to decompress
exp	address to store exponent at

Returns

significant in range [0.5, 1)

Exceptions

|--|

5.1.2.34 half_cast() [1/2]

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 halfs returns the argument unmodified.

Template Parameters

T	destination type (half or built-in arithmetic type)
U	source type (half or built-in arithmetic type)

Parameters

Returns

arg converted to destination type

Exceptions

FE_INVALID	if T is integer type and result is not representable as T
FE_OVERFLOW,UNDERFLOW,INEXACT	according to rounding

5.1.2.35 half_cast() [2/2]

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 halfs returns the argument unmodified.

Template Parameters

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

Parameters

arg	value to cast

Returns

arg converted to destination type

FE_INVALID	if T is integer type and result is not representable as T
FE_OVERFLOW,UNDERFLOW,INEXACT	according to rounding

5.1.2.36 hypot() [1/2]

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

See also: Documentation for std::hypot.

Parameters

X	first argument
У	second argument

Returns

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

Exceptions

FE_INVALID	if x or y is signaling NaN
FE_OVERFLOW,UNDERFLOW,INEXACT	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

Х	first argument
У	second argument
Z	third argument

Returns

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

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

5.1.2.38 ilogb()

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

Parameters

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

Returns

floating-point exponent

Return values

FP_ILOGB0	for zero
FP_ILOGBNAN	for NaN
INT_MAX	for infinity

Exceptions

5.1.2.39 isfinite()

Check if finite number. See also: Documentation for std::isfinite.

Parameters

```
arg number to check
```

Return values

true	if neither infinity nor NaN
false	else

5.1.2.40 isgreater()

Quiet comparison for greater than. **See also:** Documentation for std::isgreater.

X	first operand
У	second operand

Return values

true	if x greater than y
false	else

5.1.2.41 isgreaterequal()

Quiet comparison for greater equal. **See also:** Documentation for std::isgreaterequal.

Parameters

X	first operand
У	second operand

Return values

true	if x greater equal y
false	else

5.1.2.42 isinf()

Check for infinity. **See also:** Documentation for std::isinf.

Parameters

arg	number to check

Return values

true	for positive or negative infinity
false	else

5.1.2.43 isless()

Quiet comparison for less than. **See also:** Documentation for std::isless.

X	first operand
У	second operand

Return values

true	if x less than y
false	else

5.1.2.44 islessequal()

Quiet comparison for less equal. See also: Documentation for std::islessequal.

Parameters

Х	first operand
У	second operand

Return values

true	if x less equal y
false	else

5.1.2.45 islessgreater()

```
\label{eq:half_constexpr} \begin{array}{c} \text{HALF\_CONSTEXPR bool half\_float::} \text{islessgreater (} \\ \text{half } x, \\ \text{half } y) \quad \text{[inline]} \end{array}
```

Quiet comarison for less or greater. **See also:** Documentation for std::islessgreater.

Parameters

X	first operand
У	second operand

Return values

true	if either less or greater
false	else

5.1.2.46 isnan()

Check for NaN. See also: Documentation for std::isnan.

arg number to check	(
---------------------	---

Return values

true	for NaNs
false	else

5.1.2.47 isnormal()

Check if normal number. **See also:** Documentation for std::isnormal.

Parameters

arg number to check

Return values

true	if normal number
false	if either subnormal, zero, infinity or NaN

5.1.2.48 isunordered()

Quiet check if unordered. **See also:** Documentation for std::isunordered.

Parameters

Х	first operand	
y second operand		

Return values

true	if unordered (one or two NaN operands)
false	else

5.1.2.49 Idexp()

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

See also: Documentation for std::ldexp.

arg	number to modify	
exp	power of two to multiply with	

Returns

arg multplied by 2 raised to exp

Exceptions

FE_INVALID	for signaling NaN
FE_OVERFLOW,UNDERFLOW,INEXACT	according to rounding

5.1.2.50 lgamma()

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

rg function argument

Returns

natural logarith of gamma function for arg

Exceptions

FE_INVALID	for signaling NaN
FE_DIVBYZERO	for 0 or negative integer arguments
FE_OVERFLOW,UNDERFLOW,INEXACT	according to rounding

5.1.2.51 log()

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

See also: Documentation for std::log.

arg	function argument
-----	-------------------

Returns

logarithm of arg to base e

Exceptions

FE_INVALID	for signaling NaN or negative argument
FE_DIVBYZERO	for 0
FE_OVERFLOW,UNDERFLOW,INEXACT	according to rounding

5.1.2.52 log10()

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

See also: Documentation for std::log10.

Parameters

arg	function argument
-----	-------------------

Returns

logarithm of arg to base 10

Exceptions

FE_INVALID	for signaling NaN or negative argument
FE_DIVBYZERO	for 0
FE_OVERFLOW,UNDERFLOW,INEXACT	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 \sim 1% of inputs for any other rounding mode.

See also: Documentation for std::log1p.

arg	function argument
-----	-------------------

Returns

logarithm of arg plus 1 to base e

Exceptions

FE_INVALID	for signaling NaN or argument <-1
FE_DIVBYZERO	for -1
FE_OVERFLOW,UNDERFLOW,INEXACT	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

```
arg function argument
```

Returns

logarithm of arg to base 2

Exceptions

FE_INVALID	for signaling NaN or negative argument
FE_DIVBYZERO	for 0
FE_OVERFLOW,UNDERFLOW,INEXACT	according to rounding

5.1.2.55 logb()

Extract exponent. **See also:** Documentation for std::logb.

Parameters

```
arg number to query
```

Returns

floating-point exponent

FE_INVALID	for signaling NaN
FE_DIVBYZERO	for 0

5.1.2.56 Irint()

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

Parameters

```
arg half expression to round
```

Returns

nearest integer using default rounding mode

Exceptions

FE_INVALID	if value is not representable as `long`
FE_INEXACT	if value had to be rounded

5.1.2.57 Iround()

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

Parameters

```
arg half to round
```

Returns

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

Exceptions

```
FE_INVALID if value is not representable as `long`
```

5.1.2.58 modf()

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

arg	number to decompress
iptr	address to store integer part at

Returns

fractional part

Exceptions

```
FE_INVALID for signaling NaN
```

5.1.2.59 nanh()

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

Parameters

```
arg 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

```
arg half expression to round
```

Returns

nearest integer using default rounding mode

Exceptions

```
FE_INVALID for signaling NaN
```

5.1.2.61 nextafter()

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

from	value to compute next representable value for
to	direction towards which to compute next value

Returns

next representable value after from in direction towards to

Exceptions

FE_INVALID	for signaling NaN
FE_OVERFLOW	for infinite result from finite argument
FE_UNDERFLOW	for subnormal result

5.1.2.62 nexttoward()

Next representable value. **See also:** Documentation for std::nexttoward.

Parameters

from value to compute next representable value	
to	direction towards which to compute next value

Returns

next representable value after from in direction towards to

Exceptions

FE_INVALID	for signaling NaN
FE_OVERFLOW	for infinite result from finite argument
FE_UNDERFLOW	for subnormal result

5.1.2.63 operator"!=()

Comparison for inequality.

X	first operand
У	second operand

Return values

true	if operands not equal
false	else

Exceptions

FE_INVALID	if x or y is NaN
------------	------------------

5.1.2.64 operator*()

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

Parameters

Х	left operand
У	right operand

Returns

product of half expressions

Exceptions

FE_INVALID	if multiplying 0 with infinity or if x or y is signaling NaN
FE_OVERFLOW,UNDERFLOW,INEXACT	according to rounding

5.1.2.65 operator+() [1/2]

Identity.

Parameters

```
arg operand
```

Returns

unchanged operand

5.1.2.66 operator+() [2/2]

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

Parameters

X	left operand
у	right operand

Returns

sum of half expressions

Exceptions

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

5.1.2.67 operator-() [1/2]

Negation.

Parameters

```
arg operand
```

Returns

negated operand

5.1.2.68 operator-() [2/2]

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

Parameters

Х	left operand
У	right operand

Returns

difference of half expressions

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

5.1.2.69 operator/()

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

Parameters

X	left operand
У	right operand

Returns

quotient of half expressions

Exceptions

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

5.1.2.70 operator<()

Comparison for less than.

Parameters

X	first operand
У	second operand

Return values

true	if x less than y
false	else

$FE_INVALID$ if x or y is NaN

5.1.2.71 operator<<()

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

Parameters

out	output stream to write into
arg	half expression to write

Returns

reference to output stream

5.1.2.72 operator<=()

Comparison for less equal.

Parameters

X	first operand
У	second operand

Return values

true	if x less equal y
false	else

Exceptions

```
FE_INVALID if x or y is NaN
```

5.1.2.73 operator==()

Comparison for equality.

X	first operand
у	second operand

Return values

true	if operands equal
false	else

Exceptions

5.1.2.74 operator>()

Comparison for greater than.

Parameters

X	first operand
У	second operand

Return values

true	if x greater than y
false	else

Exceptions

FE_INVALID	if x or y is NaN
------------	------------------

5.1.2.75 operator>=()

Comparison for greater equal.

Parameters

Х	first operand
V	second operand

Return values

true	if x greater equal y	
false	else	

Exceptions

```
FE_INVALID if x or y is NaN
```

5.1.2.76 operator>>()

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

in	input stream to read from	
arg	half to read into	

Returns

reference to input stream

Exceptions

5.1.2.77 pow()

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

Х	base	
У	exponent	

Returns

x raised to y

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

5.1.2.78 remainder()

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

Parameters

X	first operand	
У	second operand	

Returns

remainder of floating-point division.

Exceptions

FE_INVALID	if x is infinite or y is 0 or if x or y is signaling NaN
------------	--

5.1.2.79 remquo()

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

Parameters

X	first operand	
У	second operand	
quo	address to store some bits of quotient at	

Returns

remainder of floating-point division.

FE_INVALID	if x is infinite or y is 0 or if x or y is signaling NaN
------------	--

5.1.2.80 rint()

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

Parameters

arg	half expression to round
-----	--------------------------

Returns

nearest integer using default rounding mode

Exceptions

FE_INVALID	for signaling NaN
FE_INEXACT	if value had to be rounded

5.1.2.81 round()

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

Parameters

```
arg half to round
```

Returns

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

Exceptions

FE_INVALID	for signaling NaN
FE_INEXACT	if value had to be rounded

5.1.2.82 rsqrt()

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.

nt

Returns

reciprocal of square root of arg

Exceptions

FE_INVALID	for signaling NaN and negative arguments
FE_INEXACT	according to rounding

5.1.2.83 scalbln()

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

See also: Documentation for std::scalbln.

Parameters

arg	number to modify
ехр	power of two to multiply with

Returns

arg multplied by 2 raised to exp

Exceptions

FE_INVALID	for signaling NaN
FE_OVERFLOW,UNDERFLOW,INEXACT	according to rounding

5.1.2.84 scalbn()

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

See also: Documentation for std::scalbn.

arg	number to modify
exp	power of two to multiply with

Returns

arg multplied by 2 raised to exp

Exceptions

FE_INVALID	for signaling NaN
FE_OVERFLOW,UNDERFLOW,INEXACT	according to rounding

5.1.2.85 signbit()

Check sign. **See also:** Documentation for std::signbit.

Parameters

arg	number to check
-----	-----------------

Return values

true	for negative number
false	for positive number

5.1.2.86 sin()

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

See also: Documentation for std::sin.

Parameters

arg	function argument

Returns

sine value of arg

FE_INVALID	for signaling NaN or infinity
FE_OVERFLOW,UNDERFLOW,INEXACT	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

arg	function argument	
sin	variable to take sine of arg	
cos	variable to take cosine of arg	

Exceptions

FE_INVALID	for signaling NaN or infinity
FE_OVERFLOW,UNDERFLOW,INEXACT	according to rounding

5.1.2.88 sinh()

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

See also: Documentation for std::sinh.

Parameters

arg function argument

Returns

hyperbolic sine value of arg

FE_INVALID	for signaling NaN
FE_OVERFLOW,UNDERFLOW,INEXACT	according to rounding

5.1.2.89 sqrt()

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

See also: Documentation for std::sqrt.

Parameters

```
arg function argument
```

Returns

square root of arg

Exceptions

FE_INVALID	for signaling NaN and negative arguments
FE_INEXACT	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

```
arg function argument
```

Returns

tangent value of arg

Exceptions

FE_INVALID	for signaling NaN or infinity
FE_OVERFLOW,UNDERFLOW,INEXACT	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.

arg	function argument
-----	-------------------

Returns

hyperbolic tangent value of arg

Exceptions

FE_INVALID	for signaling NaN
FE_OVERFLOW,UNDERFLOW,INEXACT	according to rounding

5.1.2.92 tgamma()

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

arg	function argument
-----	-------------------

Returns

gamma function value of arg

Exceptions

FE_INVALID	for signaling NaN, negative infinity or negative integer arguments
FE_DIVBYZERO	for 0
FE_OVERFLOW,UNDERFLOW,INEXACT	according to rounding

5.1.2.93 trunc()

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

Parameters

```
arg half to round
```

Returns

nearest integer not greater in magnitude than arg

FE_INVALID	for signaling NaN
FE_INEXACT	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:

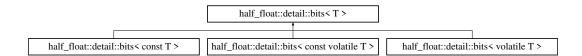
• GL_Scene/half.hpp

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

54 Class Documentation

6.2.1 Detailed Description

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

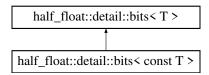
Type traits for floating-point bits.

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 >

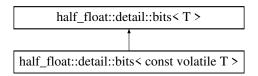
• typedef unsigned char type

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

56 Class Documentation

6.6.1 Detailed Description

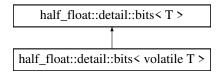
Unsigned integer of (at least) 32 bits width.

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()

Constructor de la cámara.

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

Parameters

start_position	La posición inicial de la cámara en el espacio 3D.
up_direction	La dirección "arriba" de la cámara.
start_yaw	El ángulo de yaw inicial de la cámara.
start_pitch	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()

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

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

6.9.3.3 process_mouse_movement()

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

x_offset	El cambio en la posición X del ratón.
y_offset	El cambio en la posición Y del ratón.
constraint_pitch	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.

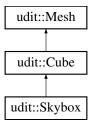
La clase Cube crea y gestiona un cubo 3D. Ofrece constructores para crear un cubo con un tamaño específico y con la opción de invertir las normales. Hereda de la clase Mesh y aprovecha sus funcionalidades para el procesamiento y renderizado del cubo en un entorno OpenGL.

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

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

```
size 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

size	El tamaño de los lados del cubo.
inverted	Si es true, 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()

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

camera 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

running	Un parámetro que indica si el bucle de la aplicación sigue en ejecución. Si se establece a false, el bucle terminará.
delta_time	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]

Constructor.

Parameters

mant	mantissa as 1.31
е	exponent

6.15.2.2 f31() [2/2]

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

Constructor.

Parameters

abs	unsigned half-precision value
abo	andighted than prodictor value

6.15.3 Friends And Related Symbol Documentation

6.15.3.1 operator*

Multiplication operator.

Parameters

а	first operand
b	second operand

Returns

a*b

6.15.3.2 operator+

Addition operator.

Parameters

а	first operand
b	second operand

Returns

a + b

6.15.3.3 operator-

Subtraction operator.

Parameters

а	first operand
b	second operand

Returns

a - b

6.15.3.4 operator/

Division operator.

Parameters

а	first operand
b	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

- $\bullet \ \ template {<} typename, typename, std::float_round_style {>}$
 - 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 Igamma (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 Irint (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 neccessarily have to be of exactly 16-bits size. But on any reasonable implementation the actual binary representation of this type will most probably not ivolve 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]

Conversion constructor.

Parameters

rhs | float to convert

Exceptions

FE OVERFLOW,UNDERFLOW,INEXACT according to re

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]

Arithmetic assignment.

Parameters

rhs single-precision value to multiply with

Returns

reference to this half

Exceptions

FE↔	according to operator=()

6.16.3.3 operator*=() [2/2]

Arithmetic assignment.

Template Parameters

```
T type of concrete half expression
```

Parameters

```
rhs half expression to multiply with
```

Returns

reference to this half

Exceptions

```
FE

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

FE⊷	according to operator+(half,half)

6.16.3.5 operator++() [2/2]

Postfix increment.

Returns

non-incremented half value

Exceptions

```
FE← according to operator+(half,half)
```

6.16.3.6 operator+=() [1/2]

Arithmetic assignment.

Parameters

rhs single-precision value to add

Returns

reference to this half

Exceptions

```
FE← according to operator=()
```

6.16.3.7 operator+=() [2/2]

Arithmetic assignment.

Template Parameters

T type of concrete half expression	1
------------------------------------	---

Parameters

rhs half ex	pression to add
-------------	-----------------

Returns

reference to this half

Exceptions

FE⊷	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

```
FE

according to operator-(half,half)

_...
```

6.16.3.9 operator--() [2/2]

Postfix decrement.

Returns

non-decremented half value

Exceptions

FE↔	according to operator-(half,half)

6.16.3.10 operator-=() [1/2]

Arithmetic assignment.

Parameters

rhs single-precision value to subtract

Returns

reference to this half

Exceptions

FE⊷	according to operator=()

6.16.3.11 operator-=() [2/2]

Arithmetic assignment.

Template Parameters

Parameters

rhs half expression to subtract

Returns

reference to this half

Exceptions

FE↔	according to operator-(half,half)

6.16.3.12 operator/=() [1/2]

Arithmetic assignment.

Parameters

rhs | single-precision value to divide by

Returns

reference to this half

Exceptions

```
FE← according to operator=()
```

6.16.3.13 operator/=() [2/2]

Arithmetic assignment.

Template Parameters



Parameters

Returns

reference to this half

Exceptions

```
FE → according to operator/(half,half)
```

6.16.3.14 operator=()

Assignment operator.

Parameters

rhs single-precision value to copy from

Returns

reference to this half

Exceptions

6.16.4 Friends And Related Symbol Documentation

6.16.4.1 acos

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

See also: Documentation for std::acos.

Parameters

arg function argument

Returns

arc cosine value of arg

Exceptions

FE_INVALID	for signaling NaN or if abs(arg) > 1
FE_OVERFLOW,UNDERFLOW,INEXACT	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

arg function argument

Returns

area cosine value of arg

Exceptions

FE_INVALID	for signaling NaN or arguments <1
FE_OVERFLOW,UNDERFLOW,INEXACT	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

arg function argument

Returns

arc sine value of arg

Exceptions

FE_INVALID	for signaling NaN or if $abs(arg) > 1$
FE_OVERFLOW,UNDERFLOW,INEXACT	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

arg function argument

Returns

area sine value of arg

Exceptions

FE_INVALID	for signaling NaN
FE_OVERFLOW,UNDERFLOW,INEXACT	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

arg	function argument
-----	-------------------

Returns

arc tangent value of arg

Exceptions

FE_INVALID	for signaling NaN
FE_OVERFLOW,UNDERFLOW,INEXACT	according to rounding

6.16.4.6 atan2

```
half atan2 (  \label{eq:half y, half x, half x}  \mbox{half } x) \quad \mbox{[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

У	numerator
X	denominator

Returns

arc tangent value

Exceptions

FE_INVALID	if x or y is signaling NaN
FE_OVERFLOW,UNDERFLOW,INEXACT	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

arg	function argument
-----	-------------------

Returns

area tangent value of arg

Exceptions

FE_INVALID	for signaling NaN or if abs(arg) > 1
FE_DIVBYZERO	for +/-1
FE_OVERFLOW,UNDERFLOW,INEXACT	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

arg	function argument
-----	-------------------

Returns

cubic root of arg

Exceptions

FE_INVALID	for signaling NaN
FE_INEXACT	according to rounding

6.16.4.9 ceil

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

Parameters

arg	half to round

Returns

nearest integer not less than arg

Exceptions

FE_INVALID	for signaling NaN
FE_INEXACT	if value had to be rounded

6.16.4.10 copysign

```
HALF_CONSTEXPR half copysign ( \label{eq:half_x} \text{half } x \text{,} \text{half } y) \quad [\text{friend}]
```

Take sign. **See also:** Documentation for std::copysign.

Parameters

X	value to change sign for
У	value to take sign from

Returns

value equal to x in magnitude and to y in sign

6.16.4.11 cos

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

See also: Documentation for std::cos.

Parameters

```
arg function argument
```

Returns

cosine value of arg

Exceptions

FE_INVALID	for signaling NaN or infinity
FE_OVERFLOW,UNDERFLOW,INEXACT	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

arg	function argument
-----	-------------------

Returns

hyperbolic cosine value of arg

Exceptions

FE_INVALID	for signaling NaN
FE_OVERFLOW,UNDERFLOW,INEXACT	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

	arg	function argument
--	-----	-------------------

Returns

error function value of arg

Exceptions

FE_INVALID	for signaling NaN
FE_OVERFLOW,UNDERFLOW,INEXACT	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

arg	function argument	
arg	function argumen	t

Returns

1 minus error function value of arg

Exceptions

FE_INVALID	for signaling NaN
FE_OVERFLOW,UNDERFLOW,INEXACT	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

```
arg function argument
```

Returns

e raised to arg

Exceptions

FE_INVALID	for signaling NaN
FE_OVERFLOW,UNDERFLOW,INEXACT	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

arg function argument

Returns

2 raised to arg

Exceptions

FE_INVALID	for signaling NaN
FE_OVERFLOW,UNDERFLOW,INEXACT	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::expml.

Parameters

arg	function argument
-----	-------------------

Returns

e raised to arg and subtracted by 1

Exceptions

FE_INVALID	for signaling NaN
FE_OVERFLOW,UNDERFLOW,INEXACT	according to rounding

6.16.4.18 fabs

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

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

Parameters

```
arg operand
```

Returns

absolute value of arg

6.16.4.19 fdim

```
half fdim (  \label{eq:half x, half x, half y}  \mbox{half } y) \quad \mbox{[friend]}
```

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

See also: Documentation for std::fdim.

Parameters

X	first operand
У	second operand

Returns

x - y or 0 if difference negative

Exceptions

FE↔	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

arg	half to round
-----	---------------

Returns

nearest integer not greater than arg

Exceptions

FE_INVALID	for signaling NaN
FE_INEXACT	if value had to be rounded

6.16.4.21 fma

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

See also: Documentation for std::fma.

Parameters

X	first operand
У	second operand
Z	third operand

Returns

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

Exceptions

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

6.16.4.22 fmax

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

Parameters

X	first operand
У	second operand

Returns

maximum of operands, ignoring quiet NaNs

Exceptions

```
FE_INVALID if x or y is signaling NaN
```

6.16.4.23 fmin

Minimum of half expressions. **See also:** Documentation for std::fmin.

Parameters

X	first operand
У	second operand

Returns

minimum of operands, ignoring quiet NaNs

Exceptions

```
FE_INVALID if x or y is signaling NaN
```

6.16.4.24 fmod

```
half fmod (  \label{eq:half x, half x, half y}  \mbox{half } y) \quad \mbox{[friend]}
```

Remainder of division. **See also:** Documentation for std::fmod.

Parameters

X	first operand
у	second operand

Returns

remainder of floating-point division.

Exceptions

FE_INVALID	if x is infinite or y is 0 or if x or y is signaling NaN
------------	--

6.16.4.25 fpclassify

```
\begin{array}{c} {\tt HALF\_CONSTEXPR} \ \ {\tt int} \ \ {\tt fpclassify} \ \ (\\ {\tt half} \ \ {\tt arg}) \ \ \ [{\tt friend}] \end{array}
```

Classify floating-point value. **See also:** Documentation for std::fpclassify.

Parameters

arg number to classify

Return values

FP_ZERO	for positive and negative zero
FP_SUBNORMAL	for subnormal numbers
FP_INFINITY	for positive and negative infinity
FP_NAN	for NaNs
FP_NORMAL	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

arg	number to decompress
exp	address to store exponent at

Returns

significant in range [0.5, 1)

Exceptions

FE_INVALID	for signaling NaN
------------	-------------------

6.16.4.27 hypot [1/2]

```
half hypot (  \label{eq:half x, half x, half y}  \mbox{ [friend]}
```

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

See also: Documentation for std::hypot.

Parameters

Х	first argument
У	second argument

Returns

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

Exceptions

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

6.16.4.28 hypot [2/2]

```
half hypot (  \begin{array}{cccc} & \text{half } x, \\ & \text{half } y, \\ & \text{half } z) & \text{[friend]} \end{array}
```

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

See also: Documentation for std::hypot.

Parameters

X	first argument
у	second argument
Z	third argument

Returns

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

Exceptions

	FE_INVALID	if x, y or z is signaling NaN
FE_OVERFLOW,UNDE	RFLOW,INEXACT	according to rounding of the final square root

6.16.4.29 ilogb

Extract exponent. See also: Documentation for $\mbox{ std::ilogb.}$

Parameters

arg	number to query

Returns

floating-point exponent

Return values

FP_ILOGB0	for zero
FP_ILOGBNAN	for NaN
INT_MAX	for infinity

Exceptions

FE_INVALID	for 0 or infinite values
------------	--------------------------

6.16.4.30 isfinite

```
\begin{array}{c} {\tt HALF\_CONSTEXPR~bool~isfinite~(} \\ {\tt half~arg)} & [{\tt friend}] \end{array}
```

Check if finite number. **See also:** Documentation for std::isfinite.

Parameters

arg	number to check
-----	-----------------

Return values

true	if neither infinity nor NaN
false	else

6.16.4.31 isgreater

```
\label{eq:half_constexpr} \begin{array}{c} \text{HALF\_CONSTEXPR bool isgreater (} \\ & \text{half } x, \\ & \text{half } y) \quad \text{[friend]} \end{array}
```

Quiet comparison for greater than. **See also:** Documentation for std::isgreater.

Parameters

X	first operand
У	second operand

Return values

true	if x greater than y
false	else

6.16.4.32 isgreaterequal

```
\label{eq:half_constexpr} \begin{array}{c} \text{HALF\_CONSTEXPR bool isgreaterequal (} \\ & \text{half } x, \\ & \text{half } y) \quad [\text{friend}] \end{array}
```

Quiet comparison for greater equal. **See also:** Documentation for std::isgreaterequal.

Parameters

X	first operand
У	second operand

Return values

true	if x greater equal y
false	else

6.16.4.33 isinf

Check for infinity. **See also:** Documentation for std::isinf.

Parameters

ara	number to check
u, g	mannoon to onlook

Return values

true	for positive or negative infinity
false	else

6.16.4.34 isless

```
\label{eq:half_constexpr} \begin{array}{c} \text{HALF\_CONSTEXPR bool isless (} \\ & \text{half } x, \\ & \text{half } y) \quad \text{[friend]} \end{array}
```

Quiet comparison for less than. See also: Documentation for std::isless.

Parameters

Х	first operand
у	second operand

Return values

true	if x less than y
false	else

6.16.4.35 islessequal

```
\begin{array}{c} {\tt HALF\_CONSTEXPR~bool~islessequal~(}\\ {\tt half~x,}\\ {\tt half~y)~[friend]} \end{array}
```

Quiet comparison for less equal. **See also:** Documentation for std::islessequal.

Parameters

X	first operand
У	second operand

Return values

true	if x less equal y
false	else

6.16.4.36 islessgreater

Quiet comarison for less or greater. **See also:** Documentation for std::islessgreater.

Parameters

X	first operand
У	second operand

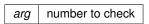
Return values

true	if either less or greater
false	else

6.16.4.37 isnan

Check for NaN. **See also:** Documentation for std::isnan.

Parameters



Return values

true	for NaNs
false	else

6.16.4.38 isnormal

```
\begin{array}{c} {\tt HALF\_CONSTEXPR~bool~isnormal~(} \\ {\tt half~arg)} & {\tt [friend]} \end{array}
```

Check if normal number. **See also:** Documentation for std::isnormal.

arg	number to check
-----	-----------------

Return values

true	if normal number
false	if either subnormal, zero, infinity or NaN

6.16.4.39 Igamma

```
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

arg	function argument
-----	-------------------

Returns

natural logarith of gamma function for arg

Exceptions

FE_INVALID	for signaling NaN
FE_DIVBYZERO	for 0 or negative integer arguments
FE_OVERFLOW,UNDERFLOW,INEXACT	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

arg	function argument

Returns

logarithm of arg to base e

Exceptions

FE_INVALID	for signaling NaN or negative argument
FE_DIVBYZERO	for 0
FE_OVERFLOW,UNDERFLOW,INEXACT	according to rounding

6.16.4.41 log10

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

See also: Documentation for std::log10.

Parameters

arg function argument

Returns

logarithm of arg to base 10

Exceptions

FE_INVALID	for signaling NaN or negative argument
FE_DIVBYZERO	for 0
FE_OVERFLOW,UNDERFLOW,INEXACT	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 \sim 1% of inputs for any other rounding mode.

See also: Documentation for std::log1p.

Parameters

arg function argument

Returns

logarithm of arg plus 1 to base e

Exceptions

FE_INVALID	for signaling NaN or argument $<$ -1
FE_DIVBYZERO	for -1
FE_OVERFLOW,UNDERFLOW,INEXACT	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> f	unction argument
--------------	------------------

Returns

logarithm of arg to base 2

Exceptions

FE_INVALID	for signaling NaN or negative argument
FE_DIVBYZERO	for 0
FE_OVERFLOW,UNDERFLOW,INEXACT	according to rounding

6.16.4.44 logb

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

Extract exponent. **See also:** Documentation for std::logb.

Parameters

```
arg number to query
```

Returns

floating-point exponent

Exceptions

FE_INVALID	for signaling NaN
FE_DIVBYZERO	for 0

6.16.4.45 Irint

```
long lrint (
          half arg) [friend]
```

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

Parameters

arg	half expression to round
-----	--------------------------

Returns

nearest integer using default rounding mode

Exceptions

FE_INVALID	if value is not representable as `long`	
FE_INEXACT	VEXACT if value had to be rounded	

6.16.4.46 Iround

```
long lround (
          half arg) [friend]
```

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

Parameters

```
arg half to round
```

Returns

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

Exceptions

FE_INVALID	if value is not representable as `long`
------------	---

6.16.4.47 modf

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

arg	number to decompress
iptr	address to store integer part at

Returns

fractional part

Exceptions

```
FE_INVALID for signaling NaN
```

6.16.4.48 nanh

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

Parameters

```
arg 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

```
arg half expression to round
```

Returns

nearest integer using default rounding mode

Exceptions

```
FE_INVALID for signaling NaN
```

6.16.4.50 nextafter

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

Parameters

from	value to compute next representable value for
to	direction towards which to compute next value

Returns

next representable value after from in direction towards to

Exceptions

FE_INVALID	for signaling NaN
FE_OVERFLOW	for infinite result from finite argument
FE_UNDERFLOW	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

from	value to compute next representable value for
to	direction towards which to compute next value

Returns

next representable value after from in direction towards to

Exceptions

FE_INVALID	for signaling NaN
FE_OVERFLOW	for infinite result from finite argument
FE_UNDERFLOW	for subnormal result

6.16.4.52 operator"!=

Comparison for inequality.

X	first operand
У	second operand

Return values

true	if operands not equal	
false	else	

Exceptions

FE_INVALID	if x or y is NaN
------------	------------------

6.16.4.53 operator*

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

Parameters

X	left operand
У	right operand

Returns

product of half expressions

Exceptions

FE_INVALID	if multiplying 0 with infinity or if x or y is signaling NaN
FE_OVERFLOW,UNDERFLOW,INEXACT	according to rounding

6.16.4.54 operator+

```
half operator+ (  \label{eq:half x, half x, half y}  \mbox{half } y) \quad \mbox{[friend]}
```

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

Parameters

Х	left operand
У	right operand

Returns

sum of half expressions

Exceptions

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

6.16.4.55 operator- [1/2]

Negation.

Parameters

arg operand

Returns

negated operand

6.16.4.56 operator- [2/2]

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

Parameters

X	left operand
У	right operand

Returns

difference of half expressions

Exceptions

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

6.16.4.57 operator/

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

X	left operand
У	right operand

Returns

quotient of half expressions

Exceptions

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

6.16.4.58 operator<

```
\label{eq:half_constexpr_noerr} \begin{array}{ll} \text{HALF\_CONSTEXPR\_NOERR bool operator} < \text{ (} \\ & \text{half } x\text{,} \\ & \text{half } y\text{)} \quad \text{[friend]} \end{array}
```

Comparison for less than.

Parameters

X	first operand
У	second operand

Return values

true	if x less than y
false	else

Exceptions

```
FE_INVALID if x or y is NaN
```

6.16.4.59 operator <<

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

Parameters

out	output stream to write into
arg	half expression to write

Returns

reference to output stream

6.16.4.60 operator<=

```
\label{eq:half_constexpr_noem} \begin{split} \text{HALF\_CONSTEXPR\_NOERR bool operator} <= & ( \\ & \text{half } x, \\ & \text{half } y) \quad \text{[friend]} \end{split}
```

Comparison for less equal.

Parameters

Х	first operand
У	second operand

Return values

true	if x less equal y
false	else

Exceptions

```
FE_INVALID if x or y is NaN
```

6.16.4.61 operator==

```
\label{eq:half_constexpr_noer} \begin{array}{l} \text{HALF\_CONSTEXPR\_NOERR bool operator} == \\ & \text{half } x, \\ & \text{half } y) \quad [\text{friend}] \end{array}
```

Comparison for equality.

Parameters

X	first operand
У	second operand

Return values

true	if operands equal
false	else

Exceptions

FE_INVALID	if x or y is NaN
------------	------------------

6.16.4.62 operator>

```
\label{eq:half_constexpr_noerr} \begin{array}{l} \text{HALF\_CONSTEXPR\_NOERR bool operator} > \text{ (} \\ & \text{half } x, \\ & \text{half } y) \quad \text{[friend]} \end{array}
```

Comparison for greater than.

Parameters

X	first operand	
У	second operand	

Return values

true	if x greater than y	
false	else	

Exceptions

FE INVALID	if x or y is NaN
------------	------------------

6.16.4.63 operator>=

```
\label{eq:half_constexpr_noem} \begin{split} \text{HALF\_CONSTEXPR\_NOERR bool operator>= (} \\ & \text{half } x, \\ & \text{half } y) \quad [\text{friend}] \end{split}
```

Comparison for greater equal.

Parameters

X	first operand	
У	second operand	

Return values

true	if x greater equal y
false	else

Exceptions

```
FE_INVALID if x or y is NaN
```

6.16.4.64 operator>>

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

in	input stream to read from
arg half to read into	

Returns

reference to input stream

Exceptions

FE_OVERFLOW,UNDERFLOW,INEXACT	according to rounding
-------------------------------	-----------------------

6.16.4.65 pow

```
half pow (  \label{eq:half x, half x, half y}  \mbox{half } y) \quad \mbox{[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

X	base
У	exponent

Returns

x raised to y

Exceptions

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

6.16.4.66 remainder

```
half remainder (  \label{eq:half x, half x, half y}  \mbox{half } y) \quad \mbox{[friend]}
```

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

Parameters

X	first operand
У	second operand

Returns

remainder of floating-point division.

Exceptions

FE_INVALID	if x is infinite or y is 0 or if x or y is signaling NaN
------------	--

6.16.4.67 remquo

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

Parameters

X	first operand
У	second operand
quo	address to store some bits of quotient at

Returns

remainder of floating-point division.

Exceptions

FE_INVALID	if x is infinite or y is 0 or if x or y 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

arg	half expression to round
-----	--------------------------

Returns

nearest integer using default rounding mode

Exceptions

FE_INVALID	for signaling NaN
FE_INEXACT	if value had to be rounded

6.16.4.69 round

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

Parameters

```
arg half to round
```

Returns

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

Exceptions

FE_INVALID	for signaling NaN
FE_INEXACT	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.

arg	function argument
-----	-------------------

Returns

reciprocal of square root of arg

Exceptions

FE_INVALID	for signaling NaN and negative arguments
FE_INEXACT	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

arg	number to modify
ехр	power of two to multiply with

Returns

arg multplied by 2 raised to exp

Exceptions

FE_INVALID	for signaling NaN
FE_OVERFLOW,UNDERFLOW,INEXACT	according to rounding

6.16.4.72 signbit

```
\begin{array}{c} {\tt HALF\_CONSTEXPR~bool~signbit~(} \\ {\tt half~arg)} & [{\tt friend}] \end{array}
```

Check sign. **See also:** Documentation for std::signbit.

Parameters

arg	number to check

Return values

true	for negative number
false	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

arg	function argument
-----	-------------------

Returns

sine value of arg

Exceptions

FE_INVALID	for signaling NaN or infinity
FE_OVERFLOW,UNDERFLOW,INEXACT	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

arg	function argument
sin	variable to take sine of arg
cos	variable to take cosine of arg

Exceptions

FE_INVALID	for signaling NaN or infinity
FE_OVERFLOW,UNDERFLOW,INEXACT	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

```
arg function argument
```

Returns

hyperbolic sine value of arg

Exceptions

FE_INVALID	for signaling NaN
FE_OVERFLOW,UNDERFLOW,INEXACT	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

```
arg function argument
```

Returns

square root of arg

Exceptions

FE_INVALID	for signaling NaN and negative arguments
FE_INEXACT	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

arg	function argument
-----	-------------------

Returns

tangent value of arg

Exceptions

FE_INVALID	for signaling NaN or infinity
FE_OVERFLOW,UNDERFLOW,INEXACT	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

arg	function argument
-----	-------------------

Returns

hyperbolic tangent value of arg

Exceptions

FE_INVALID	for signaling NaN
FE_OVERFLOW,UNDERFLOW,INEXACT	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.

arg	function argument
-----	-------------------

Returns

gamma function value of arg

Exceptions

FE_INVALID	for signaling NaN, negative infinity or negative integer arguments
FE_DIVBYZERO	for 0
FE_OVERFLOW,UNDERFLOW,INEXACT	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

arg half to round	
-------------------	--

Returns

nearest integer not greater in magnitude than arg

Exceptions

FE_INVALID	for signaling NaN
FE_INEXACT	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

 $template < typename\ T,\ typename\ U,\ std::float_round_style\ R = (std::float_round_style)(HALF_ROUND_{\leftarrow}\ STYLE) >$

```
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

T	destination type
U	source type
R	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:

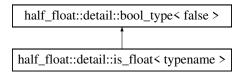
GL_Scene/half.hpp

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 >

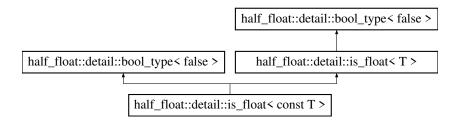
Type traits for floating-point types.

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 >:

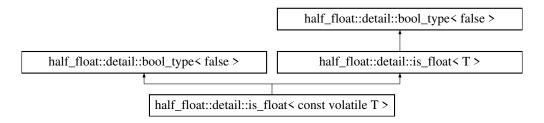


The documentation for this struct was generated from the following file:

· 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 >:

```
half_float::detail::bool_type< false > half_float::detail::bool_type< true > 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 >:

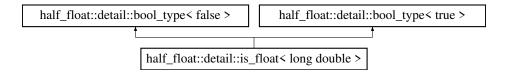
```
half_float::detail::bool_type< false > half_float::detail::bool_type< true > half_float::detail::is_float< float >
```

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 >:

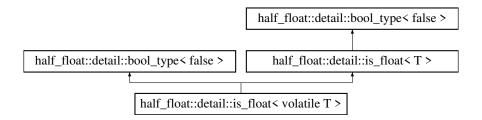


The documentation for this struct was generated from the following file:

· 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()

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

pos	Posición de la luz en el espacio 3D.
col	Color de la luz, especificado en formato RGB.
ambient	Intensidad de la luz ambiental.
diffuse	Intensidad de la luz difusa.

6.28.3 Member Function Documentation

6.28.3.1 make_light()

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.

6.29 Mesh Class Reference 117

Parameters

pos	Posición de la luz en el espacio 3D.
col	Color de la luz, especificado en formato RGB.
ambient	Intensidad de la luz ambiental.
diffuse	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()

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

program⊷	El identificador del programa de shader de OpenGL.
_id	

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

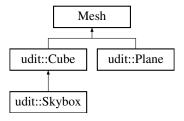
- GL_Scene/Light.hpp
- · GL_Scene/Light.cpp

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 Mesh Class Reference 119

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()

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

```
path 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()

Crea los VBOs y el VAO necesarios para la malla.

Parameters

mesh name	Nombre de la malla a crear.
micon name	i voimbre de la mana 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()

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

type	Tipo de malla a crear.
path	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()

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.

6.29 Mesh Class Reference 121

Parameters

view_matrix	Matriz de vista.
-------------	------------------

6.29.3.7 resize()

Ajusta la matriz de proyección.

Establece la matriz de proyección en el shader para la correcta visualización.

Parameters

projection_matrix	Matriz de proyección.
-------------------	-----------------------

6.29.3.8 rotate()

Rota la malla.

Aplica una rotación a la matriz de transformación de la malla.

Parameters

rotation	Eje de rotación.
angle	Ángulo de rotación en grados.

6.29.3.9 scale()

Escala la malla.

Aplica una escala a la matriz de transformación de la malla.

Parameters

scale	Factor de escala.

6.29.3.10 set_mesh_type()

Establece el tipo de malla.

Parameters

6.29.3.11 set_model_view_matrix()

Establece la matriz de transformación del modelo.

Parameters

matrix Nueva matriz de transformación del modelo.

6.29.3.12 set_shader()

Asocia un shader a la malla.

Permite asociar un shader para ser usado al renderizar la malla.

Parameters

shader Puntero al shader a asociar.

6.29.3.13 translate()

Realiza una traslación de la malla.

Aplica una traslación a la matriz de transformación de la malla.

Parameters

translation Vector de traslación.

6.29.3.14 update()

```
void udit::Mesh::update () [virtual]
```

Actualiza la malla.

Función de actualizacion 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:

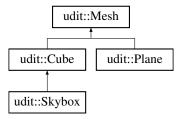
- GL_Scene/Mesh.hpp
- · GL Scene/Mesh.cpp

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()

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.

path 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()

Crea los VBOs y el VAO necesarios para la malla.

Parameters

```
mesh_name 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()

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

type	Tipo de malla a crear.	
path	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()

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

view_matrix	Matriz de vista.
-------------	------------------

6.30.3.7 resize()

Ajusta la matriz de proyección.

Establece la matriz de proyección en el shader para la correcta visualización.

projection matrix	Matriz de proyección.

6.30.3.8 rotate()

Rota la malla.

Aplica una rotación a la matriz de transformación de la malla.

Parameters

rotation	Eje de rotación.
angle	Ángulo de rotación en grados.

6.30.3.9 scale()

Escala la malla.

Aplica una escala a la matriz de transformación de la malla.

Parameters

scal	e	Factor de escala.	

6.30.3.10 set_mesh_type()

Establece el tipo de malla.

Parameters

type	Tipo de malla.

6.30.3.11 set_model_view_matrix()

Establece la matriz de transformación del modelo.

Parameters

matrix Nueva matriz de transformación del modelo.

6.30.3.12 set_shader()

Asocia un shader a la malla.

Permite asociar un shader para ser usado al renderizar la malla.

Parameters

6.30.3.13 translate()

Realiza una traslación de la malla.

Aplica una traslación a la matriz de transformación de la malla.

Parameters

```
translation Vector de traslación.
```

6.30.3.14 update()

```
void udit::Mesh::update () [virtual]
```

Actualiza la malla.

Función de actualizacion 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.

6.34 Plane Class Reference 133

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]

Constructor que define el tamaño del plano.

Crea un plano cuadrado con el tamaño especificado.

Parameters

```
size 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

width	Ancho del plano.
height	Alto del plano.
columns Número de columnas del plano (resolución horizo	
rows	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]

Constructor que define el tamaño del plano.

Crea un plano cuadrado con el tamaño especificado.

Parameters

size	Tamaño del plano en ambas dimensiones (ancho y alto).
------	---

6.35.2.3 Plane() [3/3]

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

width	Ancho del plano.
height	Alto del plano.
columns Número de columnas del plano (resolución horizor	
rows	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 137

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()

Constructor de la escena.

Constructor.

Inicializa una nueva escena con el ancho y alto especificados.

Parameters

width	Ancho de la ventana de renderizado.
height	Alto de la ventana de renderizado.

Inicializa una escena

Parameters

width	Ancho de la escena
height	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

l	width	Nuevo ancho de la ventana.
	height	Nuevo alto de la ventana.

6.36.3.3 set_lights()

Establece las luces en el shader.

Configura las luces de la escena dentro del shader, enviando los parámetros necesarios al programa de sombreado.

Parameters

shader_program←	Identificador del programa de sombreado (shader).
_id	

6.36.3.4 set_projection_matrix()

Establece la matriz de proyección para la cámara.

Establece la matriz de proyección que será usada para renderizar la escena.

Parameters

projection	Matriz de proyección.
------------	-----------------------

6.36.3.5 set_view_matrix()

Establece la matriz de vista para la cámara.

Establece la matriz de vista que será usada para renderizar la escena.

Parameters

view 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

width	Ancho de la ventana de renderizado.
height	Alto de la ventana de renderizado.

Inicializa una escena

Parameters

width	Ancho de la escena
height	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

width	Nuevo ancho de la ventana.
height	Nuevo alto de la ventana.

6.37.3.3 set lights()

Establece las luces en el shader.

Configura las luces de la escena dentro del shader, enviando los parámetros necesarios al programa de sombreado.

Parameters

shader_program←	Identificador del programa de sombreado (shader).
_id	

6.37.3.4 set_projection_matrix()

Establece la matriz de proyección para la cámara.

Establece la matriz de proyección que será usada para renderizar la escena.

Parameters

projection Matriz de proyección.

6.37.3.5 set_view_matrix()

Establece la matriz de vista para la cámara.

Establece la matriz de vista que será usada para renderizar la escena.

Parameters

view 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::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]

Constructor para crear un shader con tipos y fuentes especificadas.

Parameters

type	Tipo de shader (e.g., SKYBOX, GEOMETRY).	
vertex_source	Código fuente para el vertex shader.	
fragment_source	Código fuente para el fragment shader.	
name	Nombre del shader.	

6.38.2.3 ∼Shader()

```
udit::Shader::\simShader ()
```

Destructor.

Libera los recursos asociados al shader.

6.38.3 Member Function Documentation

6.38.3.1 compile_shaders()

Compila los shaders.

Compilador de los shaders construidos.

Compila un vertex shader y un fragment shader usando el código fuente proporcionado.

Parameters

vertex_shader_code	Código fuente del vertex shader.
fragment_shader_code	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

type	Tipo de shader.	
vertex_shader	Código fuente del vertex shader.	
fragment_shader	Código fuente del fragment shader.	
texture_paths	Rutas a las texturas asociadas.	
name	Nombre del shader.	

Returns

Objeto Shader creado.

6.38.3.9 set_name()

Establece el nombre del shader.

Parameters

name	Nombre del shader.
------	--------------------

6.38.3.10 set_texture()

Establece una textura para el shader.

Parameters

texture Puntero a la textura que será asignada
--

6.38.3.11 set_texture_scale()

Establece la escala de las texturas asociadas al shader.

Parameters

scale	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.

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]

Constructor para crear un shader con tipos y fuentes especificadas.

Parameters

type	Tipo de shader (e.g., SKYBOX, GEOMETRY).	
vertex_source	Código fuente para el vertex shader.	
fragment_source	Código fuente para el fragment shader.	
name	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()

Compila los shaders.

Compilador de los shaders construidos.

Compila un vertex shader y un fragment shader usando el código fuente proporcionado.

Parameters

vertex_shader_code	Código fuente del vertex shader.
fragment_shader_code	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

type	Tipo de shader.
vertex_shader	Código fuente del vertex shader.
fragment_shader	Código fuente del fragment shader.
texture_paths	Rutas a las texturas asociadas.
name	Nombre del shader.

Returns

Objeto Shader creado.

6.39.3.9 set_name()

Establece el nombre del shader.

Parameters

name Nombre del shader.

6.39.3.10 set_texture()

Establece una textura para el shader.

Parameters

	texture	Puntero a la textura que será asignada al shader.
--	---------	---

6.39.3.11 set_texture_scale()

Establece la escala de las texturas asociadas al shader.

Parameters

scale	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:

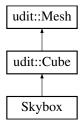
- GL_Scene/Shader.hpp
- GL_Scene/Shader.cpp

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< 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.

6.40.1 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]

Constructor que permite especificar el tamaño y las texturas del skybox.

Parameters

size	Tamaño del cubo que representará el skybox.
faces	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:

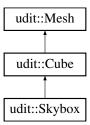
- GL Scene/Skybox.hpp
- GL_Scene/Skybox.cpp

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]

Constructor que permite especificar el tamaño y las texturas del skybox.

Parameters

size	Tamaño del cubo que representará el skybox.
faces	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.
- \sim 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()

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

path	Ruta al archivo de la textura (imagen).
texture_unit	Unidad de textura (GL_TEXTURE0, GL_TEXTURE1, etc.).
type	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()

Establece el tipo de la textura (COLOR o HEIGHT).

Parameters

```
type 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()

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

path	Ruta al archivo de la textura (imagen).	
texture_unit	Unidad de textura (GL_TEXTURE0, GL_TEXTURE1, etc.).	
type	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()

Establece el tipo de la textura (COLOR o HEIGHT).

Parameters

```
type 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_ \leftarrow 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.

• \sim 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()

Constructor de la ventana.

Parameters

title	Titulo de la ventana
left_x	Posicion de la ventana en el eje x
top_y	Posicion de la ventana en el eje y
width	Ancho de la ventana
height	Alto de la ventana
context_details	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** { $UNDEFINED = SDL_WINDOWPOS_UNDEFINED$, $CENTERED = SDL_WINDOWPOS_$ \leftarrow 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
- \sim 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()

Constructor de la ventana.

Parameters

title	Titulo de la ventana
left_x	Posicion de la ventana en el eje x
top_y	Posicion de la ventana en el eje y
width	Ancho de la ventana
height	Alto de la ventana
context_details	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.
00007
00008 #pragma once
00009
00010 #include "glm.hpp"
00011 #include <qtc/matrix_transform.hpp>
00012 #include <gtc/constants.hpp>
00014 enum class CameraMovement
00015 {
00016
          FORWARD.
00017
          BACKWARD,
00018
          LEFT,
00019
          RIGHT,
00020
          UP,
00021
          DOWN
00022 };
00033 class Camera
00034 {
00035 public:
00041
         glm::vec3 position;
00042
00049
          glm::vec3 front;
00050
00056
          glm::vec3 up;
00057
00064
          glm::vec3 right;
00065
00072
          glm::vec3 world_up;
00073
00079
          float yaw;
00080
00086
          float pitch;
00087
00093
          float movement_speed;
00094
00100
          float mouse_sensitivity;
00101
00107
00108
00119
          Camera(glm::vec3 start_position, glm::vec3 up_direction, float start_yaw, float start_pitch);
00120
          glm::mat4 get_view_matrix() const;
00129
00130
          void process_keyboard(CameraMovement direction, float delta_time);
00141
00152
          void process_mouse_movement(float x_offset, float y_offset, bool constraint_pitch = true);
00153
00154 private:
00161
          void update_camera_vectors();
00162 };
```

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
          private:
00026
00032
              float size;
00033
00034
        public:
00040
             Cube();
00041
              Cube (bool inverted);
00050
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 //
00004 //
00005 //
          Created by Alonso García on 23/12/24.
00006 //
00007
00008 #pragma once
00009
00010 #include "SDL.h"
00010 #INClude SDL.N"
00011 #include "glm.hpp"
00012 #include "Camera.hpp"
00022 class EventHandler
00023 {
00024 public:
          EventHandler(Camera& camera)
00034
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 >
```

Namespaces

- · namespace half float
- · namespace std

Extensions to the C++ standard library.

 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 >

Macros

- #define HALF_GCC_VERSION (__GNUC__*100+__GNUC_MINOR__)
- #define HALF ICC VERSION 0
- #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)
- $\bullet \;\; template {<} std:: float_round_style \; R, \, unsigned \; int \; F, \, bool \; S, \, bool \; N, \, bool \; I {>} \\$
 - unsigned int half_float::detail::fixed2half (uint32 m, int exp=14, unsigned int sign=0, int s=0)
- $\bullet \;\; template < std::float_round_style \; R >$
- 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)
- $\bullet \ \ template < std::float_round_style \ R, \ typename \ T >$
- 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)
- $\bullet \;\; template {<} std:: float_round_style \; R, \; bool \; E, \; bool \; I, \; typename \; T {>} \\$
 - 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::sgrt (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

    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>

Thalf 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 intruction 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 halfs 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:

std::float_round_style	value	rounding
std::round_indeterminate	-1	fastest
std::round_toward_zero	0	toward zero
std::round_to_nearest	1	to nearest (default)
std::round_toward_infinity	2	toward positive infinity
std::round_toward_neg_infinity	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 <cfloat> 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 $\mathtt{HUGE_VAL}[F|L]$ from $<\mathtt{cmath}>$ 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()

Reduce argument for trigonometric functions.

Parameters

abs	half-precision floating-point value
k	value to take quarter period

Returns

abs reduced to [-pi/4,pi/4] as Q0.30

7.4.3.2 area()

Area function and postprocessing. This computes the value directly in Q2.30 using the representation $asinh|acosh(x)| = log(x+sqrt(x^2+|-1))$.

Template Parameters

R	rounding mode to use	
S	true for asinh, false for acosh	

Parameters

Returns

asinh|acosh(arg) converted to half-precision

Exceptions

FE_OVERFLOW	on overflows
FE_UNDERFLOW	on underflows
FE_INEXACT	if no other exception occurred

7.4.3.3 arithmetic_shift()

Platform-independent arithmetic right shift.

Parameters

arg	integer value in two's complement
i	shift amount (at most 31)

Returns

arg right shifted for i bits with possible sign extension

7.4.3.4 atan2()

Fixed point arc tangent. This uses the CORDIC algorithm in vectoring mode.

Parameters

	ту	y coordinate as Q0.30	
Ī	mx x coordinate as Q0.30		
Γ	n	number of iterations (at most 31)	

Returns

arc tangent of my / mx as Q1.30

7.4.3.5 atan2_args()

Get arguments for atan2 function.

Parameters

abs	half-precision floating-point value
-----	-------------------------------------

Returns

abs and sqrt(1 - $abs^{\wedge}2$) as Q0.30

7.4.3.6 builtin_isinf()

Check for infinity.

Template Parameters

T argument type (builtin floating-point type)

Parameters

arg	value to query

Return values

true	if infinity
false	else

7.4.3.7 builtin_isnan()

Check for NaN.

Template Parameters

Parameters

arg	value to query
-----	----------------

Return values

true	if not a number
false	else

7.4.3.8 builtin_signbit()

Check sign.

Template Parameters

```
T | argument type (builtin floating-point type)
```

Parameters

Return values

true	if signbit set	
false	else	

7.4.3.9 check_underflow()

Check value for underflow.

Parameters

arg	non-zero half-precision value to check
-----	--

Returns

arg

Exceptions

```
FE_UNDERFLOW if arg is subnormal
```

7.4.3.10 compsignal()

Check and signal for any NaN.

Parameters

	first half-precision value to check	
У	second half-precision value to check	

Return values

true	if either x or y is NaN
false	else

Exceptions

```
FE_INVALID if x or y is NaN
```

7.4.3.11 divide64()

64-bit division.

Parameters

X	upper 32 bit of dividend	
У	divisor	
s	variable to store sticky bit for rounding	

Returns

7.4.3.12 erf()

Error function and postprocessing. This computes the value directly in Q1.31 using the approximations given here.

Template Parameters

R	rounding mode to use
С	true for comlementary error function, false else

Parameters

arg	half-precision function argument
-----	----------------------------------

Returns

approximated value of error function in half-precision

Exceptions

FE_OVERFLOW	on overflows
FE_UNDERFLOW	on underflows
FE_INEXACT	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()

Fixed point binary exponential. This uses the BKM algorithm in E-mode.

Parameters

m	exponent in [0,1) as Q0.31
n	number of iterations (at most 32)

Returns

```
2^{\ \ m} as Q1.31
```

7.4.3.15 exp2_post()

Postprocessing for binary exponential.

Template Parameters

R rounding mode	to use
-----------------	--------

Parameters

m	fractional part of as Q0.31
exp	absolute value of unbiased exponent
esign	sign of actual exponent
sign	sign bit of result
n	number of BKM iterations (at most 32)

Returns

value converted to half-precision

Exceptions

FE_OVERFLOW	on overflows
FE_UNDERFLOW	on underflows
FE_INEXACT	if value had to be rounded or / is `true`

7.4.3.16 fixed2half()

Convert fixed point to half-precision floating-point.

Template Parameters

R	rounding mode to use
F	number of fractional bits in [11,31]
S	true for signed, false for unsigned
Ν	true for additional normalization step, false if already normalized to 1.F
1	true to always raise INEXACT exception, false to raise only for rounded results

Parameters

m	mantissa in Q1.F fixed point format
exp	biased exponent - 1
sign	half-precision value with sign bit only
s	sticky bit (or of all but the most significant already discarded bits)

Returns

value converted to half-precision

Exceptions

FE_OVERFLOW	on overflows
FE_UNDERFLOW	on underflows
FE_INEXACT	if value had to be rounded or / is `true`

7.4.3.17 float2half()

Convert floating-point to half-precision.

Template Parameters

R	rounding mode to use
T	source type (builtin floating-point type)

Parameters

value	floating-point value to convert
-------	---------------------------------

Returns

rounded half-precision value

Exceptions

FE_OVERFLOW	on overflows
FE_UNDERFLOW	on underflows
FE_INEXACT	if value had to be rounded

7.4.3.18 float2half_impl() [1/3]

Convert IEEE double-precision to half-precision.

Template Parameters

R	rounding mode to use	

Parameters

value	double-precision value to convert
-------	-----------------------------------

Returns

rounded half-precision value

Exceptions

FE_OVERFLOW	on overflows
FE_UNDERFLOW	on underflows
FE_INEXACT	if value had to be rounded

7.4.3.19 float2half_impl() [2/3]

Convert IEEE single-precision to half-precision. Credit for this goes to Jeroen van der Zijp.

Template Parameters

```
R rounding mode to use
```

Parameters

value single-precision value to convert

Returns

rounded half-precision value

Exceptions

FE_OVERFLOW	on overflows
FE_UNDERFLOW	on underflows
FE_INEXACT	if value had to be rounded

7.4.3.20 float2half_impl() [3/3]

Convert non-IEEE floating-point to half-precision.

Template Parameters

R	rounding mode to use
Т	source type (builtin floating-point type)

Parameters

value	floating-point value to convert
-------	---------------------------------

Returns

rounded half-precision value

Exceptions

FE_OVERFLOW	on overflows
FE_UNDERFLOW	on underflows
FE_INEXACT	if value had to be rounded

7.4.3.21 gamma()

Gamma function and postprocessing. This approximates the value of either the gamma function or its logarithm directly in Q1.31.

Template Parameters

R	rounding mode to use
L	true for lograithm of gamma function, false for gamma function

Parameters

arg	half-precision floating-point value
-----	-------------------------------------

Returns

Igamma/tgamma(arg) in half-precision

Exceptions

FE_OVERFLOW	on overflows
FE_UNDERFLOW	on underflows
FE_INEXACT	if arg is not a positive integer

7.4.3.22 half2float()

Convert half-precision to floating-point.

Template Parameters

Parameters

value half-precision value to conve	ert
-------------------------------------	-----

Returns

floating-point value

7.4.3.23 half2float_impl() [1/3]

Convert half-precision to IEEE double-precision.

Parameters

value half-precision value to c	convert
---------------------------------	---------

Returns

double-precision value

7.4.3.24 half2float_impl() [2/3]

Convert half-precision to IEEE single-precision. Credit for this goes to Jeroen van der Zijp.

Parameters

value	half-precision value to convert
-------	---------------------------------

Returns

single-precision value

7.4.3.25 half2float_impl() [3/3]

Convert half-precision to non-IEEE floating-point.

Template Parameters

T type to convert to (builtin integer to	type)
--	-------

Parameters

ert
֡

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

R	rounding mode to use	
E	true for round to even, false for round away from zero	
1	true to raise INEXACT exception (if inexact), false to never raise it	
T	type to convert to (buitlin integer type with at least 16 bits precision, excluding any implicit sign bits)	

Parameters

rt
֡

Returns

rounded integer value

Exceptions

FE_INVALID	if value is not representable in type T
FE_INEXACT	if value had to be rounded and / is `true`

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

abs	half-precision floating-point value	
ехр	exp variable to take unbiased exponent of larger result	
n	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()

Hypotenuse square root and postprocessing.

Template Parameters

R	rounding mode to use
---	----------------------

Parameters

r	mantissa as Q2.30	
exp	biased exponent	

Returns

square root converted to half-precision

Exceptions

FE_OVERFLOW	on overflows
FE_UNDERFLOW	on underflows
FE_INEXACT	if value had to be rounded

7.4.3.29 int2half()

Convert integer to half-precision floating-point.

Template Parameters

R	rounding mode to use	
T type to convert (builtin integer type		

Parameters

lue integral value to convert	value
-------------------------------	-------

Returns

rounded half-precision value

Exceptions

FE_OVERFLOW	on overflows
FE_INEXACT	if value had to be rounded

7.4.3.30 integral()

Round half-precision number to nearest integer value.

Template Parameters

R	rounding mode to use	
Ε	true for round to even, false for round away from zero	
1	true to raise INEXACT exception (if inexact), false to never raise it	

Parameters

value	half-precision value to round
	man production rando to round

Returns

half-precision bits for nearest integral value

Exceptions

FE_INVALID	for signaling NaN
FE_INEXACT	if value had to be rounded and / is `true`

7.4.3.31 invalid()

 $\verb|HALF_CONSTEXPR_NOERR| unsigned int half_float::detail::invalid () \\ [inline]$

Raise domain error and return NaN. return quiet NaN

Exceptions

```
FE INVALID
```

7.4.3.32 log2()

Fixed point binary logarithm. This uses the BKM algorithm in L-mode.

Parameters

m	mantissa in [1,2) as Q1.30
n	number of iterations (at most 32)

Returns

log2(m) as Q0.31

7.4.3.33 log2_post()

Postprocessing for binary logarithm.

Template Parameters

R	rounding mode to use
L	logarithm for base transformation as Q1.31

Parameters

m	fractional part of logarithm as Q0.31
ilog	signed integer part of logarithm
exp	biased exponent of result
sign	sign bit of result

Returns

value base-transformed and converted to half-precision

Exceptions

FE_OVERFLOW	on overflows
FE_UNDERFLOW	on underflows
FE_INEXACT	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

Q	true to compute full quotient, false else
R	true to compute signed remainder, false for positive remainder

Parameters

X	first operand as positive finite half-precision value
У	second operand as positive finite half-precision value
quo	adress to store quotient at, nullptr if Q false

Returns

modulus of x / y

7.4.3.35 mulhi()

upper part of 64-bit multiplication.

Template Parameters

R	rounding mode to use

Parameters

Χ	first factor
у	second factor

Returns

```
upper 32 bit of x * y
```

7.4.3.36 multiply64()

64-bit multiplication.

Parameters

Χ	first factor
У	second factor

Returns

upper 32 bit of x * y rounded to nearest

7.4.3.37 overflow()

Half-precision overflow.

Template Parameters

R rounding mode to use

Parameters

gn half-precision value with sign bit only	у
--	---

Returns

rounded overflowing half-precision value

Exceptions

```
FE_OVERFLOW
```

7.4.3.38 pole()

```
\label{eq:half_constexpr_noerr} \mbox{HALF\_CONSTEXPR\_NOERR unsigned int half\_float::detail::pole (} \\ \mbox{unsigned int } sign = 0) \mbox{ [inline]}
```

Raise pole error and return infinity.

Parameters

sign half-precision value with sign bit only	bit only
--	----------

Returns

half-precision infinity with sign of sign

Exceptions

```
FE_DIVBYZERO
```

7.4.3.39 raise()

Raise floating-point exception.

Parameters

flags	exceptions to raise
cond	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

R	rounding mode to use
1	true to always raise INEXACT exception, false to raise only for rounded results

Parameters

value	finite half-precision number to round
g	guard bit (most significant discarded bit)
s	sticky bit (or of all but the most significant discarded bits)

Returns

rounded half-precision value

Exceptions

FE_OVERFLOW	on overflows
FE_UNDERFLOW	on underflows
FE_INEXACT	if value had to be rounded or / is `true`

7.4.3.41 select()

Select value or signaling NaN.

Parameters

X	preferred half-precision value
У	ignored half-precision value except for signaling NaN

Returns

y if signaling NaN, x otherwise

Exceptions

7.4.3.42 sign_mask()

Platform-independent sign mask.

Parameters

arg	integer value in two's complement	
-----	-----------------------------------	--

Return values

-1	if <i>arg</i> negative
0	if arg positive

7.4.3.43 signal() [1/3]

Signal and silence signaling NaN.

Parameters

nan	half-precision NaN value
-----	--------------------------

Returns

quiet NaN

Exceptions

FE_INVALID	if <i>nan</i> is signaling NaN
------------	--------------------------------

7.4.3.44 signal() [2/3]

Signal and silence signaling NaNs.

Parameters

x first half-precision value to check	
у	second half-precision value to check

Returns

quiet NaN

Exceptions

```
FE_INVALID if x or y is signaling NaN
```

7.4.3.45 signal() [3/3]

```
\label{eq:half_constexpr_noerr} \begin{tabular}{ll} HALF\_CONSTEXPR\_NOERR unsigned int half\_float::detail::signal (         unsigned int <math>x,          unsigned int y,          unsigned int z) [inline]
```

Signal and silence signaling NaNs.

Parameters

X	first half-precision value to check
У	second half-precision value to check
Z	third half-precision value to check

Returns

quiet NaN

Exceptions

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

mz	angle in [-pi/2,pi/2] as Q1.30
n	number of iterations (at most 31)

Returns

sine and cosine of mz as Q1.30

7.4.3.47 sqrt()

Fixed point square root.

Template Parameters

Parameters

r	radicand in Q1.F fixed point format
ехр	exponent

Returns

square root as Q1.F/2

7.4.3.48 tangent_post()

Division and postprocessing for tangents.

Template Parameters

R	rounding mode to use
---	----------------------

Parameters

my	dividend as Q1.31
mx	divisor as Q1.31
ехр	biased exponent of result
sign	sign bit of result

Returns

quotient converted to half-precision

Exceptions

FE_OVERFLOW	on overflows
FE_UNDERFLOW	on underflows
FE_INEXACT	if no other exception occurred

7.4.3.49 underflow()

Half-precision underflow.

Template Parameters

R	rounding mode to use

Parameters

sign half-precision value with sign bit only
--

Returns

rounded underflowing half-precision value

Exceptions

FE_UNDERFLOW

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 <rauy@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
00027 #if defined(__INTEL_COMPILER)
00028
          #define HALF_ICC_VERSION __INTEL_COMPILER
00029 #elif defined(__TCC)
00030 #define HALF_ICC_VERSION _
00031 #elif defined(__ICL)
          #define HALF_ICC_VERSION __ICL
00033 #else
00034
          #define HALF_ICC_VERSION 0
00035 #endif
00036
00037 // check C++11 language features
                                                                       // clang
00038 #if defined(__clang__)
00039
          #if __has_feature(cxx_static_assert) && !defined(HALF_ENABLE_CPP11_STATIC_ASSERT)
00040
               #define HALF_ENABLE_CPP11_STATIC_ASSERT 1
00041
          #endif
          #if __has_feature(cxx_constexpr) && !defined(HALF_ENABLE_CPP11_CONSTEXPR) #define HALF_ENABLE_CPP11_CONSTEXPR 1
00042
00043
00044
          #endif
          #if __has_feature(cxx_noexcept) && !defined(HALF_ENABLE_CPP11_NOEXCEPT) #define HALF_ENABLE_CPP11_NOEXCEPT 1
00045
00046
00047
          #endif
00048
          #if __has_feature(cxx_user_literals) && !defined(HALF_ENABLE_CPP11_USER_LITERALS)
#define HALF_ENABLE_CPP11_USER_LITERALS 1
00049
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
          #if (defined(
00054
      #if (defined(__GXX_EXPERIMENTAL_CXX0X__) || __cplusplus >= 201103L) &&
!defined(HALF_ENABLE_CPP11_LONG_LONG)
00055
             #define HALF_ENABLE_CPP11_LONG_LONG 1
00056
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
00064
          #if HALF_ICC_VERSION >= 1400 && !defined(HALF_ENABLE_CPP11_CONSTEXPR)
00065
               #define HALF_ENABLE_CPP11_CONSTEXPR 1
00066
          #endif
          #if HALF_ICC_VERSION >= 1400 && !defined(HALF_ENABLE_CPP11_NOEXCEPT)
00067
               #define HALF_ENABLE_CPP11_NOEXCEPT 1
00068
00069
          #if HALF_ICC_VERSION >= 1110 && !defined(HALF_ENABLE_CPP11_STATIC_ASSERT)
00070
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
          #endif
```

7.5 half.hpp 199

```
00076 #elif defined(__GNUC
         #if defined(__GXX_EXPERIMENTAL_CXXOX__) || __cplusplus >= 201103L
00077
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)
                   #define HALF_ENABLE_CPP11_USER_LITERALS 1
00082
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)
                   #define HALF_ENABLE_CPP11_LONG_LONG 1
00094
00095
               #endif
00096
          #endif
00097
          #define HALF_TWOS_COMPLEMENT_INT 1
00098 #elif defined(_MSC_VER) // Visua
00099 #if _MSC_VER >= 1900 && !defined(HALF_ENABLE_CPP11_THREAD_LOCAL)
                                                                      // Visual C++
00100
              #define HALF_ENABLE_CPP11_THREAD_LOCAL 1
           #endif
          #if _MSC_VER >= 1900 && !defined(HALF_ENABLE_CPP11_USER_LITERALS)
00102
00103
              #define HALF_ENABLE_CPP11_USER_LITERALS 1
00104
          #endif
          #if _MSC_VER >= 1900 && !defined(HALF_ENABLE_CPP11_CONSTEXPR)
00105
00106
              #define HALF ENABLE CPP11 CONSTEXPR 1
00107
           #endif
          #if _MSC_VER >= 1900 && !defined(HALF_ENABLE_CPP11_NOEXCEPT)
00108
00109
               #define HALF_ENABLE_CPP11_NOEXCEPT 1
00110
          #endif
          #if _MSC_VER >= 1600 && !defined(HALF_ENABLE_CPP11_STATIC_ASSERT)
00111
00112
              #define HALF_ENABLE_CPP11_STATIC_ASSERT 1
00113
           #endif
          #if _MSC_VER >= 1310 && !defined(HALF_ENABLE_CPP11_LONG_LONG)
#define HALF_ENABLE_CPP11_LONG_LONG 1
00114
00115
00116
          #endif
00117
          #define HALF_TWOS_COMPLEMENT_INT 1
          #define HALF_POP_WARNINGS 1
00118
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)
          #if defined(__GXX_EXPERIMENTAL_CXX0X__) || __cplusplus >= 201103
              #ifndef HALF_ENABLE_CPP11_TYPE_TRAITS
00127
00128
                  #define HALF_ENABLE_CPP11_TYPE_TRAITS 1
00129
              #endif
              #ifndef HALF_ENABLE_CPP11_CSTDINT
00130
00131
                  #define HALF ENABLE CPP11 CSTDINT 1
              #ifndef HALF_ENABLE_CPP11_CMATH
00133
00134
                  #define HALF_ENABLE_CPP11_CMATH 1
00135
               #endif
              #ifndef HALF_ENABLE_CPP11_HASH
00136
00137
                  #define HALF ENABLE CPP11 HASH 1
00138
               #endif
              #ifndef HALF_ENABLE_CPP11_CFENV
00139
00140
                  #define HALF_ENABLE_CPP11_CFENV 1
00141
              #endif
00142
          #endif
00143 #elif defined(__GLIBCXX__)
                                                                      // libstdc++
         #if defined(__GXX_EXPERIMENTAL_CXXOX__) || __cplusplus >= 201103
00144
00145
              #ifdef __clang__
00146
                       __GLIBCXX__ >= 20080606 && !defined(HALF_ENABLE_CPP11_TYPE_TRAITS)
00147
                       #define HALF_ENABLE_CPP11_TYPE_TRAITS 1
00148
                   #endif
                  #if __GLIBCXX__ >= 20080606 && !defined
    #define HALF_ENABLE_CPP11_CSTDINT 1
                                   >= 20080606 && !defined(HALF_ENABLE_CPP11_CSTDINT)
00149
00150
00151
                   #endif
00152
                                   >= 20080606 && !defined(HALF_ENABLE_CPP11_CMATH)
                   #if __GLIBCXX_
00153
                       #define HALF_ENABLE_CPP11_CMATH 1
00154
                   #endif
                   #if __GLIBCXX_
                       __GLIBCXX__ >= 20080606 && !defined(HALF_ENABLE_CPP11_HASH) #define HALF_ENABLE_CPP11_HASH 1
00155
00156
00157
                   #endif
                   #if __GLIBCXX__ >= 20080606 && !defined(HALF_ENABLE_CPP11_CFENV)
00158
00159
                       #define HALF_ENABLE_CPP11_CFENV 1
00160
                  #endif
00161
              #else
00162
                   #if HALF GCC VERSION >= 403 && !defined(HALF ENABLE CPP11 TYPE TRAITS)
```

```
#define HALF_ENABLE_CPP11_TYPE_TRAITS 1
                   #endif
00165
                   #if HALF_GCC_VERSION >= 403 && !defined(HALF_ENABLE_CPP11_CSTDINT)
00166
                      #define HALF ENABLE CPP11 CSTDINT 1
00167
                   #endif
                  #if HALF_GCC_VERSION >= 403 && !defined(HALF_ENABLE_CPP11_CMATH)
00168
                       #define HALF_ENABLE_CPP11_CMATH 1
00169
00170
                   #endif
00171
                   #if HALF_GCC_VERSION >= 403 && !defined(HALF_ENABLE_CPP11_HASH)
00172
                       #define HALF_ENABLE_CPP11_HASH 1
                   #endif
00173
                   #if HALF_GCC_VERSION >= 403 && !defined(HALF_ENABLE_CPP11_CFENV)
00174
00175
                       #define HALF_ENABLE_CPP11_CFENV 1
00176
00177
              #endif
00178
          #endif
00179 #elif defined(_CPPLIB_VER) // Dinkum
00180 #if _CPPLIB_VER >= 520 && !defined(HALF_ENABLE_CPP11_TYPE_TRAITS)
00181 #define HALF_ENABLE_CPP11_TYPE_TRAITS 1
                                                                       // Dinkumware/Visual C++
00182
          #endif
00183
          #if _CPPLIB_VER >= 520 && !defined(HALF_ENABLE_CPP11_CSTDINT)
00184
                   #define HALF_ENABLE_CPP11_CSTDINT 1
          #endif
00185
          #if _CPPLIB_VER >= 520 && !defined(HALF_ENABLE_CPP11_HASH)
    #define HALF_ENABLE_CPP11_HASH 1
00186
00187
          #if _CPPLIB_VER >= 610 && !defined(HALF_ENABLE_CPP11_CMATH)
00189
00190
              #define HALF_ENABLE_CPP11_CMATH 1
00191
           #endif
          #if _CPPLIB_VER >= 610 && !defined(HALF_ENABLE_CPP11_CFENV)
    #define HALF_ENABLE_CPP11_CFENV 1
00192
00193
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
          #define HALF_UNUSED_NOERR(name) name
00208
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
           #define HALF_CONSTEXPR_CONST
                                                 constexpr
          #if HALF_ERRHANDLING
00217
00218
              #define HALF_CONSTEXPR_NOERR
00219
          #else
             #define HALF_CONSTEXPR_NOERR
00220
                                                 constexpr
          #endif
00221
00222 #else
      #define HALF_CONSTEXPR
00223
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
       #define HALF_NOEXCEPT
00233
00234
          #define HALF_NOTHROW
                                   throw()
00235 #endif
00236
00237 // support thread storage
00238 #if HALF_ENABLE_CPP11_THREAD_LOCAL
          #define HALF_THREAD_LOCAL thread_local
00239
00240 #else
          #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
         #include <type_traits>
00255
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
        #define HALF_ENABLE_F16C_INTRINSICS ___F16C__
00279 #endif
00280 #if HALF_ENABLE_F16C_INTRINSICS
00281
         #include <immintrin.h>
00282 #endif
00283
00284 #ifdef HALF_DOXYGEN_ONLY
00290 #define HALF_ARITHMETIC_TYPE (undefined)
00291
00295 #define HALF_ERRHANDLING_FLAGS 0
00296
00302 #define HALF_ERRHANDLING_ERRNO 0
00310 #define HALF_ERRHANDLING_FENV
00311
00315 #define HALF_ERRHANDLING_THROW_INVALID
                                                (undefined)
00316
00320 #define HALF ERRHANDLING THROW DIVBYZERO (undefined)
00321
00325 #define HALF_ERRHANDLING_THROW_OVERFLOW
                                                 (undefined)
00326
00330 #define HALF_ERRHANDLING_THROW_UNDERFLOW
                                                (undefined)
00331
00335 #define HALF ERRHANDLING THROW INEXACT
                                                 (undefined)
00336 #endif
00337
00338 #ifndef HALF_ERRHANDLING_OVERFLOW_TO_INEXACT
00342 #define HALF_ERRHANDLING_OVERFLOW_TO_INEXACT
00343 #endif
00344
00345 #ifndef HALF_ERRHANDLING_UNDERFLOW_TO_INEXACT
00352 #define HALF_ERRHANDLING_UNDERFLOW_TO_INEXACT
00353 #endif
00354
00373 #ifndef HALF_ROUND_STYLE
         #define HALF_ROUND_STYLE 1
                                           // = std::round_to_nearest
00374
00375 #endif
00376
00382 #define HUGE_VALH std::numeric_limits<half_float::half>::infinity()
00383
00389 #define FP_FAST_FMAH
00390
00396 #define HLF ROUNDS HALF ROUND STYLE
00397
00398 #ifndef FP_ILOGB0
00399
         #define FP_ILOGB0
                                 INT_MIN
00400 #endif
00401 #ifndef FP_ILOGBNAN
         #define FP_ILOGBNAN
                                 INT MAX
00402
00403 #endif
00404 #ifndef FP_SUBNORMAL
00405
         #define FP_SUBNORMAL
00406 #endif
00407 #ifndef FP_ZERO
00408
        #define FP ZERO
                                 1
00409 #endif
00410 #ifndef FP_NAN
00411
         #define FP_NAN
00412 #endif
00413 #ifndef FP_INFINITE
00414
        #define FP_INFINITE
00415 #endif
```

```
00416 #ifndef FP_NORMAL
          #define FP_NORMAL
00417
00418 #endif
00419
00420 #if !HALF_ENABLE_CPP11_CFENV && !defined(FE_ALL_EXCEPT)
        #define FE_INVALID
#define FE_DIVBYZERO
                                   0x10
00421
                                   0x08
00423
          #define FE_OVERFLOW
                                    0x04
00424
         #define FE_UNDERFLOW
                                   0x02
00425
          #define FE INEXACT
                                   0x01
         #define FE_ALL_EXCEPT (FE_INVALID|FE_DIVBYZERO|FE_OVERFLOW|FE_UNDERFLOW|FE_INEXACT)
00426
00427 #endif
00428
00429
00432 namespace half_float
00433 {
00434
          class half:
00435
00436 #if HALF_ENABLE_CPP11_USER_LITERALS
00443
          namespace literal
00444
              half operator "" _h(long double);
00445
00446
00447 #endif
00448
00451
          namespace detail
00452
00453
          #if HALF_ENABLE_CPP11_TYPE_TRAITS
00455
               template<br/>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
              template<bool,typename T,typename> struct conditional { typedef T type; };
00466
              template<typename T, typename F> struct conditional<false, T, F> { typedef F type; };
00467
00468
00470
               template<bool> struct bool_type {};
00471
               typedef bool_type<true> true_type;
              typedef bool_type<false> false_type;
00472
00473
00475
              template<typename> struct is_float : false_type {};
00476
              template<typename T> struct is_float<const T> : is_float<T> {};
               template<typename T> struct is_float<volatile T> : is_float<T> {};
00477
00478
               template<typename T> struct is_float<const volatile T> : is_float<T> {};
00479
               template<> struct is_float<float> : true_type {};
              template<> struct is_float<double> : true_type {};
00480
              template<> struct is_float<long double> : true_type {};
00481
00482
          #endif
00483
00485
               template<typename T> struct bits { typedef unsigned char type; };
              template<typename T> struct bits<const T> : bits<T> {};
template<typename T> struct bits<volatile T> : bits<T> {};
00486
00487
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
              template<> struct bits<double> { typedef std::uint_least64_t type; };
00504
00505
          #else
00507
              typedef unsigned short uint16:
00508
00510
              typedef unsigned long uint32;
00511
00513
              typedef long int32;
00514
              template<> struct bits<float> : conditional<std::numeric_limits<unsigned</pre>
00516
      int>::digits>=32, unsigned int, unsigned long> {};
00517
00518
               #if HALF_ENABLE_CPP11_LONG_LONG
                  template<> struct bits<double> : conditional<std::numeric_limits<unsigned</pre>
00520
      long>::digits>=64,unsigned long,unsigned long long> {};
00521
             #else
00523
                  template<> struct bits<double> { typedef unsigned long type; };
00524
              #endif
00525
          #endif
00526
          #ifdef HALF ARITHMETIC TYPE
00527
00529
              typedef HALF_ARITHMETIC_TYPE internal_t;
```

```
00530
          #endif
00531
00533
              struct binary_t {};
00534
              HALF_CONSTEXPR_CONST binary_t binary = binary_t();
00536
00537
00546
              template<typename T> bool builtin_isinf(T arg)
00547
00548
              #if HALF ENABLE CPP11 CMATH
00549
                  return std::isinf(arg);
00550
              #elif defined( MSC VER)
00551
                  return !::_finite(static_cast<double>(arg)) && !::_isnan(static_cast<double>(arg));
00552
00553
                  return arg == std::numeric_limits<T>::infinity() || arg ==
      -std::numeric_limits<T>::infinity();
00554
              #endif
00555
              }
00562
              template<typename T> bool builtin_isnam(T arg)
00563
00564
              #if HALF_ENABLE_CPP11_CMATH
00565
                  return std::isnan(arg);
              #elif defined(_MSC_VER)
00566
00567
                  return ::_isnan(static_cast<double>(arg)) != 0;
00568
                  return arg != arg;
00569
00570
              #endif
00571
00572
00578
              template<typename T> bool builtin signbit(T arg)
00579
00580
               #if HALF_ENABLE_CPP11_CMATH
00581
                  return std::signbit(arg);
00582
               #else
                  return arg < T() || (arg == T() && T(1)/arg < T());</pre>
00583
              #endif
00584
00585
00586
00591
              inline uint32 sign_mask(uint32 arg)
00592
              {
                  static const int N = std::numeric_limits<uint32>::digits - 1;
00593
              #if HALF TWOS COMPLEMENT INT
00594
00595
                  return static_cast<int32>(arg) » N;
00596
00597
                  return -((arg»N)&1);
00598
              #endif
00599
00600
              inline uint32 arithmetic_shift(uint32 arg, int i)
00605
00606
00607
              #if HALF_TWOS_COMPLEMENT_INT
00608
                  return static_cast<int32>(arg) » i;
00609
                  return static_cast<int32>(arg)/(static_cast<int32>(1) «i) -
00610
      ((arg»(std::numeric limits<uint32>::digits-1))&1);
00611
              #endif
00612
00613
00617
00620
              inline int& errflags() { HALF THREAD LOCAL int flags = 0; return flags; }
00621
00625
              inline void raise(int HALF_UNUSED_NOERR(flags), bool HALF_UNUSED_NOERR(cond) = true)
00626
00627
              #if HALF ERRHANDLING
00628
                  if(!cond)
00629
              return;
#if HALF_ERRHANDLING_FLAGS
00630
00631
                  errflags() |= flags;
              #endif
00632
00633
              #if HALF_ERRHANDLING_ERRNO
00634
                  if(flags & FE_INVALID)
00635
                      errno = EDOM;
                  else if(flags & (FE_DIVBYZERO|FE_OVERFLOW|FE_UNDERFLOW))
00636
                      errno = ERANGE;
00637
00638
              #if HALF_ERRHANDLING_FENV && HALF_ENABLE_CPP11_CFENV
00639
00640
                  std::feraiseexcept(flags);
00641
              #endif
              #ifdef HALF_ERRHANDLING_THROW_INVALID
00642
00643
                  if (flags & FE INVALID)
00644
                       throw std::domain_error(HALF_ERRHANDLING_THROW_INVALID);
00645
              #endif
00646
               #ifdef HALF_ERRHANDLING_THROW_DIVBYZERO
00647
                 if(flags & FE_DIVBYZERO)
                       throw std::domain_error(HALF_ERRHANDLING_THROW_DIVBYZERO);
00648
00649
              #endif
```

```
#ifdef HALF_ERRHANDLING_THROW_OVERFLOW
                                    if(flags & FE_OVERFLOW)
00651
00652
                                              throw std::overflow_error(HALF_ERRHANDLING_THROW_OVERFLOW);
                             #endif
00653
                             #ifdef HALF_ERRHANDLING_THROW_UNDERFLOW
00654
                                     if (flags & FE_UNDERFLOW)
00655
00656
                                             throw std::underflow_error(HALF_ERRHANDLING_THROW_UNDERFLOW);
00657
                              #endif
00658
                              #ifdef HALF_ERRHANDLING_THROW_INEXACT
00659
                                     if (flags & FE INEXACT)
                                              throw std::range_error(HALF_ERRHANDLING_THROW_INEXACT);
00660
00661
                             #endif
                             #if HALF_ERRHANDLING_UNDERFLOW_TO_INEXACT
00662
00663
                                     if((flags & FE_UNDERFLOW) && !(flags & FE_INEXACT))
00664
                                             raise(FE_INEXACT);
00665
                              #endif
                              #if HALF_ERRHANDLING_OVERFLOW_TO_INEXACT
00666
                                     if((flags & FE_OVERFLOW) && !(flags & FE_INEXACT))
00667
                                              raise(FE_INEXACT);
00668
00669
                              #endif
00670
                              #endif
00671
00672
00679
                             inline HALF CONSTEXPR NOERR bool compsignal (unsigned int x, unsigned int y)
00680
00681
                              #if HALF ERRHANDLING
00682
                                      raise(FE_INVALID, (x&0x7FFF)>0x7C00 || (y&0x7FFF)>0x7C00);
00683
                              #endif
00684
                                     return (x&0x7FFF) > 0x7C00 || (y&0x7FFF) > 0x7C00;
00685
00686
00691
                             inline HALF_CONSTEXPR_NOERR unsigned int signal(unsigned int nan)
00692
00693
                              #if HALF_ERRHANDLING
00694
                                     raise(FE_INVALID, !(nan&0x200));
                              #endif
00695
00696
                                     return nan | 0x200;
00697
00698
00704
                             inline {\tt HALF\_CONSTEXPR\_NOERR} unsigned int signal (unsigned int x, unsigned int y)
00705
00706
                              #if HALF ERRHANDLING
                                     raise(FE_INVALID, ((x&0x7FFF)>0x7C00 && !(x&0x200))) || ((y&0x7FFF)>0x7C00 && !(y&0x200)));
00707
00708
                              #endif
00709
                                     return ((x&0x7FFF)>0x7C00) ? (x|0x200) : (y|0x200);
00710
00711
00718
                              \text{inline HALF\_CONSTEXPR\_NOERR unsigned int } \textbf{signal} \\  (\text{unsigned int } \textbf{x, unsigned int } \textbf{y, unsigned int } \textbf{y}, \\  \text{unsigned int } \textbf{y, unsigned int } \textbf{y}, \\  \text{unsigned int } \textbf{y, unsigned int } \textbf{y}, \\  \text{unsigned int } \textbf{y}, \\  \text{unsi
           z)
00719
00720
                             #if HALF_ERRHANDLING
                                      raise(FE_INVALID, ((x&0x7FFF)>0x7C00 && !(x&0x200)) || ((y&0x7FFF)>0x7C00 && !(y&0x200))
00721
            || ((z&0x7FFF)>0x7C00 && !(z&0x200)));
00722
                             #endif
00723
                                     return ((x&0x7FFF)>0x7C00) ? (x|0x200) : ((y&0x7FFF)>0x7C00) ? (y|0x200) : (z|0x200);
00724
00725
                             inline HALF_CONSTEXPR_NOERR unsigned int select(unsigned int \mathbf{x}, unsigned int
00731
            HALF_UNUSED_NOERR(y))
00732
00733
                              #if HALF ERRHANDLING
                                     return (((y&0x7FFF)>0x7C00) && !(y&0x200)) ? signal(y) : x;
00734
00735
                              #else
00736
                                     return x;
00737
                              #endif
00738
                             }
00739
00743
                             inline HALF CONSTEXPR NOERR unsigned int invalid()
00744
00745
                              #if HALF ERRHANDLING
00746
                                      raise(FE_INVALID);
00747
                              #endif
00748
                                     return 0x7FFF;
                             }
00749
00750
00755
                              inline HALF_CONSTEXPR_NOERR unsigned int pole(unsigned int sign = 0)
00756
                              #if HALF_ERRHANDLING
00757
00758
                                     raise(FE_DIVBYZERO);
00759
                              #endif
00760
                                    return sign | 0x7C00;
00761
                             }
00762
00767
                             inline HALF_CONSTEXPR_NOERR unsigned int <a href="mailto:check_underflow">check_underflow</a> (unsigned int arg)
00768
00769
                              #if HALF_ERRHANDLING && !HALF_ERRHANDLING_UNDERFLOW_TO_INEXACT
00770
                                      raise(FE_UNDERFLOW, !(arg&0x7C00));
```

```
00771
              #endif
                 return arg;
00772
00773
00774
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);
              #endif
00788
00789
                           (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
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);
               #endif
00804
                  return (R==std::round_toward_infinity) ? (sign+1-(sign»15)) :
00805
00806
                           (R==std::round_toward_neg_infinity) ? (sign+(sign*15)) :
00807
                           sign;
00808
              }
nngng
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
                                (R==std::round_to_nearest) ? (g&(s|value)) :
00824
                                (R==std::round\_toward\_infinity) ? (\sim(value)) & (g|s) :
00825
                                (R = std::round\_toward\_neg\_infinity) \ ? \ ((value > 15) \& (g | s)) \ : \ 0;
                   if((value\&0x7C00) == 0x7C00)
00826
                       raise(FE_OVERFLOW);
00827
                   else if (value & 0x7C00)
00828
00829
                       raise(FE_INEXACT, I || (g|s)!=0);
00830
00831
                       raise (FE_UNDERFLOW, ! (HALF_ERRHANDLING_UNDERFLOW_TO_INEXACT) | | I | | (q|s)!=0);
00832
                   return value;
00833
               #else
00834
                  return (R==std::round_to_nearest) ? (value+(g&(s|value))) :
                            (R==std::round_toward_infinity) ? (value+(~(value»15)&(g|s))) :
00835
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)
                       return (abs>0x7C00) ? signal(value) : value;
00861
00862
                  unsigned int exp = 25 - (abs * 10), mask = (1 exp) - 1;
00863
                   raise(FE_INEXACT, I && (value&mask));
                                (R==std::round_to_nearest) ? ((1«(exp-1))-(~(value»exp)&E)) :
00864
                   return ((
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
              template<std::float_round_style R,unsigned int F,bool S,bool N,bool I> unsigned int
00884
      fixed2half(uint32 m, int exp = 14, unsigned int sign = 0, int s = 0)
00885
              {
00886
00887
                       uint32 msign = sign_mask(m);
m = (m^msign) - msign;
00888
00889
                       sign = msign & 0x8000;
00890
00891
00892
00893
                       for(; m<(static_cast<uint32>(1) «F) && exp; m«=1,--exp);
00894
                   else if(exp < 0)</pre>
      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));
00895
```

```
return rounded<R, I>(sign+(exp<10)+(m>(F-10)), (m>(F-11))&1,
      s \mid ((m&((static cast < uint 32 > (1) < (F-11)) - 1))! = 0));
00897
00898
00907
              template<std::float round style R> unsigned int float2half impl(float value, true type)
00908
00909
               #if HALF ENABLE F16C INTRINSICS
                   return _mm_cvtsi128_si32(_mm_cvtps_ph(_mm_set_ss(value),
00910
                       (R==std::round_to_nearest) ? _MM_FROUND_TO_NEAREST_INT : (R==std::round_toward_zero) ? _MM_FROUND_TO_ZERO :
00911
00912
                       (R==std::round_toward_infinity) ? _MM_FROUND_TO_POS_INF :
00913
00914
                       (R==std::round_toward_neg_infinity) ? _MM_FROUND_TO_NEG_INF :
00915
                       _MM_FROUND_CUR_DIRECTION));
00916
00917
                  bits<float>::type fbits;
00918
                   std::memcpy(&fbits, &value, sizeof(float));
              #if 1
00919
00920
                   unsigned int sign = (fbits»16) & 0x8000;
                   fbits &= 0x7FFFFFFF;
00921
00922
                   if(fbits >= 0x7F800000)
00923
                       return sign | 0x7C00 | ((fbits>0x7F800000) ? (0x200|((fbits»13)&0x3FF)) : 0);
00924
                   if(fbits >= 0x47800000)
00925
                       return overflow<R>(sign);
                   if(fbits >= 0x38800000)
00926
                       return rounded<R, false>(sign|(((fbits>23)-112) < 10)|((fbits>13) & 0x3FF), (fbits>12) & 1,
00927
      (fbits&0xFFF) !=0);
00928
                   if(fbits >= 0x33000000)
00929
                       int i = 125 - (fbits * 23);
00930
                       fbits = (fbits&0x7FFFFF) | 0x800000;
00931
                       return rounded<R, false>(sign|(fbits»(i+1)), (fbits»i)&1,
00932
      (fbits&((static_cast<uint32>(1) «i)-1))!=0);
00933
00934
                   if(fbits != 0)
00935
                       return underflow<R>(sign);
00936
                  return sign;
00937
              #else
00938
                  static const uint16 base_table[512] = {
00939
                       0x0000, 0x0000,
      0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
00940
                       0x0000,\ 0x0000,
      0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
                       0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000. 0x0000.
00941
      0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
                       0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
      0x0000, 0x0000,
                       0x0000, 0x0000, 0x0000, 0x0000,
00943
                       0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
      0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
00944
                       0x0000, 0x0000,
      0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
00945
                       0x0000,
                               0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0002, 0x0004,
                       0x0020, 0x0040, 0x0080, 0x0100,
      0x0008, 0x0010,
00946
                       0x0200,
                               0x0400,
                                       0x0800, 0x0C00, 0x1000, 0x1400, 0x1800, 0x1C00, 0x2000, 0x2400,
                                       0x3800,
      0x2800, 0x2C00,
                       0x3000,
                               0x3400,
                                               0x3C00,
00947
                       0x4000, 0x4400,
                                       0x4800, 0x4C00, 0x5000, 0x5400, 0x5800, 0x5C00, 0x6000, 0x6400,
      0x6800, 0x6C00,
                       0x7000, 0x7400,
                                       0x7800, 0x7BFF,
                       0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF,
      0x7BFF, 0x7BFF,
                       0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF,
                       0x7BFF, 0x7BFF,
                                       0x7BFF, 0x7BFF,
                                                        0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF,
00949
      0x7BFF, 0x7BFF,
                       0x7BFF, 0x7BFF,
                                       0x7BFF, 0x7BFF,
00950
                       0x7BFF, 0x7BFF,
                                       0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF,
      0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF,
00951
                       0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF,
      0x7BFF, 0x7BFF,
                       0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF,
                       0x7BFF,
00952
                                               0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF,
                               0x7BFF,
                                       0x7BFF,
      0x7BFF, 0x7BFF,
                       0x7BFF, 0x7BFF,
                                       0x7BFF, 0x7BFF,
00953
                       0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF,
      0x7BFF, 0x7BFF,
                       0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF,
                       0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF,
00954
      0x7BFF, 0x7BFF,
                       0x7BFF, 0x7BFF, 0x7BFF, 0x7C00,
                       0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000,
00955
      0x8000, 0x8000,
                       0x8000,
                               0x8000,
                                       0x8000, 0x8000,
                       0x8000,
00956
                               0x8000,
                                       0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000,
      0x8000, 0x8000,
                       0x8000, 0x8000,
                                       0x8000, 0x8000,
00957
                       0x8000,
                               0x8000,
                                       0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000,
      0x8000, 0x8000,
                       0x8000, 0x8000, 0x8000, 0x8000,
00958
                       0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000,
      0x8000, 0x8000,
                       0x8000,
                               0x8000, 0x8000, 0x8000,
00959
                       0x8000.
                               0x8000,
                                       0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000,
      0x8000. 0x8000.
                       0x8000, 0x8000, 0x8000, 0x8000,
00960
                       0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000,
      0x8000, 0x8000,
                       0x8000, 0x8000, 0x8000, 0x8000,
                       0x8000,
                               0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8001, 0x8002, 0x8004,
                                       0x8080,
      0x8008, 0x8010,
                       0x8020,
                               0x8040,
                                               0x8100,
00962
                       0x8200, 0x8400, 0x8800, 0x8C00, 0x9000, 0x9400, 0x9800, 0x9C00, 0xA000, 0xA400,
      0xA800, 0xAC00, 0xB000, 0xB400, 0xB800, 0xBC00,
00963
                       0xC000, 0xC400, 0xC800, 0xCC00, 0xD000, 0xD400, 0xD800, 0xDC00, 0xE000, 0xE400,
```

```
0xE800, 0xEC00, 0xF000, 0xF400, 0xF800, 0xFBFF,
                  OxfBFF, OxfBFF, OxfBFF, OxfBFF, OxfBFF, OxfBFF, OxfBFF, OxfBFF, OxfBFF,
00964
     OxFBFF, OxFBFF, OxFBFF, OxFBFF, OxFBFF,
00965
                  OxfBFF, OxfBFF, OxfBFF, OxfBFF, OxfBFF, OxfBFF, OxfBFF, OxfBFF, OxfBFF, OxfBFF,
     OxFBFF, OxFBFF, OxFBFF, OxFBFF, OxFBFF,
00966
                  OXFBFF, OXFBFF, OXFBFF, OXFBFF, OXFBFF, OXFBFF, OXFBFF, OXFBFF, OXFBFF,
     OxFBFF, OxFBFF, OxFBFF, OxFBFF, OxFBFF,
00967
                  0xfBff, 0xfBff, 0xfBff, 0xfBff, 0xfBff, 0xfBff, 0xfBff, 0xfBff, 0xfBff, 0xfBff,
     OxFBFF, OxFBFF, OxFBFF, OxFBFF, OxFBFF,
00968
                  OxFBFF, OxFBFF, OxFBFF, OxFBFF, OxFBFF, OxFBFF, OxFBFF, OxFBFF, OxFBFF,
     OxFBFF, OxFBFF, OxFBFF, OxFBFF, OxFBFF,
00969
                  OxFBFF, OxFBFF, OxFBFF, OxFBFF, OxFBFF, OxFBFF, OxFBFF, OxFBFF, OxFBFF, OxFBFF,
    OxFBFF, OxFBFF, OxFBFF, OxFBFF, OxFBFF,
                  OxfBFF, OxfBFF, OxfBFF, OxfBFF, OxfBFF, OxfBFF, OxfBFF, OxfBFF, OxfBFF, OxfBFF,
00970
     0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFC00 };
00971
             static const unsigned char shift_table[256] = {
                  00972
                  25, 25, 25, 25, 25, 25, 25,
    25, 25, 25, 25,
00973
                  00974
                  25, 25, 25, 25,
                  25, 25, 25, 25, 25, 25, 25,
00975
                  25, 25, 25, 25, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 13, 13, 13,
    13, 13, 13, 13,
                 13, 13, 13, 13, 13, 13, 13,
00976
                  00977
                  00978
                  00979
                  24, 24, 24, 24, 24, 24, 24, 24, 24, 13 };

int sexp = fbits » 23, exp = sexp & 0xFF, i = shift_table[exp];
00980
00981
              fbits \&= 0x7FFFFF;
00982
              uint32 m = (fbits|((exp!=0) < 23)) & -static_cast < uint32 > (exp!=0 xFF);
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
00995
           template<std::float_round_style R> unsigned int float2half_impl(double value, true_type)
00996
00997
           #if HALF_ENABLE_F16C_INTRINSICS
00998
              if(R == std::round_indeterminate)
00999
                  return _mm_cvtsi128_si32(_mm_cvtps_ph(_mm_cvtpd_ps(_mm_set_sd(value)),
    _MM_FROUND_CUR_DIRECTION));
01000
           #endif
01001
              bits<double>::type dbits;
              std::memcpy(&dbits, &value, sizeof(double));
01002
              uint32 hi = dbits » 32, lo = dbits & 0xFFFFFFF;
01003
              unsigned int sign = (hi»16) & 0x8000;
01004
01005
              hi &= 0x7FFFFFFF;
01006
              if(hi >= 0x7FF00000)
                  return sign | 0x7C00 | ((dbits&0xFFFFFFFFFF) ? (0x200|((hi»10)&0x3FF)) : 0);
01007
01008
              if(hi >= 0x40F00000)
01009
                  return overflow<R>(sign);
01010
              if(hi >= 0x3F100000)
                  return rounded<R, false>(sign|(((hi»20)-1008) <10)|((hi»10)&0x3FF), (hi»9)&1,</pre>
     ((hi&0x1FF)|lo)!=0);
01012
              if(hi >= 0x3E600000)
01013
              {
01014
                  int i = 1018 - (hi \approx 20);
                  hi = (hi&0xFFFFF) | 0x100000;
01015
01016
                  return rounded<R, false>(sign|(hi»(i+1)), (hi»i)&1,
    ((hi&((static_cast<uint32>(1) «i)-1))|lo)!=0);
01017
01018
               if((hi|lo) != 0)
01019
                 return underflow<R>(sign);
01020
              return sign;
01021
01022
01031
           template<std::float_round_style R,typename T> unsigned int float2half_impl(T value, ...)
01032
01033
              unsigned int hbits = static cast < unsigned > (builtin signbit (value)) « 15;
              if(value == T())
01034
01035
                  return hbits;
01036
               if(builtin_isnan(value))
01037
                 return hbits | 0x7FFF;
              if (builtin_isinf(value))
01038
01039
                 return hbits | 0x7C00;
              int exp;
01040
01041
              std::frexp(value, &exp);
01042
              if(exp > 16)
01043
                  return overflow<R>(hbits);
01044
              if(exp < -13)
01045
                  value = std::ldexp(value, 25);
```

```
else
01047
01048
                                              value = std::ldexp(value, 12-exp);
01049
                                            hbits |= ((exp+13) \times 10);
01050
01051
                                     T ival, frac = std::modf(value, &ival);
01052
                                     int m = std::abs(static_cast<int>(ival));
                                     return rounded<R, false>(hbits+(m>1), m&1, frac!=T());
01053
01054
01055
01064
                             template<std::float_round_style R, typename T> unsigned int float2half(T value)
01065
                                      return float2half_impl<R> (value,
01066
           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;</pre>
                                     if(!value)
01079
01080
                                              return bits;
01081
                                     if(bits)
01082
                                            value = -value;
                                     if(value > 0xFFFF)
01083
                                              return overflow<R>(bits);
01084
                                     unsigned int m = static_cast<unsigned int>(value), exp = 24;
01085
                                     for(; m<0x400; m<=1,--exp);</pre>
01086
01087
                                     for(; m>0x7FF; m>=1,++exp);
01088
                                     bits |= (exp \ll 10) + m;
                                     return (exp>24) ? rounded<R,false>(bits, (value»(exp-25))&1, (((1«(exp-25))-1)&value)!=0)
01089
            : bits;
01090
                             }
01091
01096
                             inline float half2float_impl(unsigned int value, float, true_type)
01097
                             #if HALF ENABLE F16C INTRINSICS
01098
01099
                                     return _mm_cvtss_f32(_mm_cvtph_ps(_mm_cvtsi32_si128(value)));
01100
01101
                             #if 0
                                     bits<float>::type fbits = static_cast<bits<float>::type>(value&0x8000) « 16;
01102
01103
                                     int abs = value & 0x7FFF;
01104
                                     if (abs)
01105
                                     {
                                              fbits |= 0x38000000 « static_cast<unsigned>(abs>=0x7C00);
01106
                                              for(; abs<0x400; abs<=1,fbits-=0x800000);</pre>
01107
01108
                                              fbits += static_cast<bits<float>::type>(abs) « 13;
01109
01110
                             #else
                                     static const bits<float>::type mantissa_table[2048] = {
01111
                                              0x00000000, 0x33800000, 0x34000000, 0x34400000, 0x34800000, 0x34A00000, 0x34C00000,
01112
             0x34E000000,\ 0x350000000,\ 0x351000000,\ 0x352000000,\ 0x353000000,\ 0x354000000,\ 0x355000000,\ 0x356000000,
             0x35700000,
01113
                                              0 \\ x \\ 35800000, \ 0 \\ x \\ 35880000, \ 0 \\ x \\ 35900000, \ 0 \\ x \\ 35980000, \ 0 \\ x \\ 35400000, \ 0 \\ x \\ 35480000, \ 0 \\ x \\ 35800000, \ 0 \\ x \\ 358000000, \ 0 \\ x \\ 35800000, \ 0 \\ x \\ 358000000, \ 0 \\ x \\ 35800000000
             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
             0x365C0000, 0x36600000, 0x36640000, 0x36680000, 0x366C0000, 0x36700000, 0x36740000, 0x36780000,
             0x367C0000,
                                              0x36800000, 0x36820000, 0x36840000, 0x36860000, 0x36880000, 0x368A0000, 0x368C0000,
01116
             01117
                                              0x36A00000, 0x36A20000, 0x36A40000, 0x36A60000, 0x36A80000, 0x36AA0000, 0x36AC0000,
             0x36BE0000,
                                              0x36C00000, 0x36C20000, 0x36C40000, 0x36C60000, 0x36C80000, 0x36CA0000, 0x36CC0000,
01118
             0x36DE0000,
                                              0x36E00000, 0x36E20000, 0x36E40000, 0x36E60000, 0x36E80000, 0x36EA0000, 0x36EC0000,
01119
             0x36EE0000, 0x36F00000, 0x36F20000, 0x36F40000, 0x36F60000, 0x36F80000, 0x36FA0000, 0x36FC0000,
             0x36FE0000,
                                              0x37000000, 0x37010000, 0x37020000, 0x37030000, 0x37040000, 0x37050000, 0x37060000,
01120
             0x37070000, 0x37080000, 0x37090000, 0x370A0000, 0x370B0000, 0x370C0000, 0x370D0000, 0x370E0000,
                                              0x37100000, 0x37110000, 0x37120000, 0x37130000, 0x37140000, 0x37150000, 0x37160000,
01121
             0 \times 37170000, \ 0 \times 37180000, \ 0 \times 37190000, \ 0 \times 37180000, \ 0 \times 37180000, \ 0 \times 37160000, \ 0 \times 371600000, \ 0 \times 37160000, \ 0 \times 37160000, \ 0 \times 37160000, \ 0 \times 371600000, \ 0 \times 37160000, \ 0 \times 371600000, \ 0 \times 37160000, \ 0 \times 371600000, \ 0 \times 371600000, \ 0 \times 371600000, \ 0 \times 3716000000, \ 0 \times 3716000000000, \ 0 \times 37160000000
             0x371F0000,
01122
                                              0 \times 37200000, 0 \times 37210000, 0 \times 37220000, 0 \times 37230000, 0 \times 37240000, 0 \times 37250000, 0 \times 37260000,
             0x372F0000,
                                              0x37370000, 0x37380000, 0x37390000, 0x373A0000, 0x373B0000, 0x373C0000, 0x373D0000, 0x373E0000,
             0x373F0000,
                                              0 \times 37400000, 0 \times 37410000, 0 \times 37420000, 0 \times 37430000, 0 \times 37440000, 0 \times 37450000, 0 \times 37460000,
01124
             0x37470000, 0x37480000, 0x37490000, 0x37400000, 0x374B0000, 0x374C0000, 0x374D0000, 0x374E0000,
```

	0x374F0000,	
01125	0x37570000, 0x375F0000,	0x37500000, 0x37510000, 0x37520000, 0x37530000, 0x37540000, 0x37550000, 0x37560000, 0x37580000, 0x37590000, 0x37590000, 0x37500000, 0x37500000, 0x37500000, 0x37500000, 0x37500000, 0x37500000,
01126		0x37600000, 0x37610000, 0x37620000, 0x37630000, 0x37640000, 0x37650000, 0x37660000, 0x37680000, 0x37690000, 0x376A0000, 0x376B0000, 0x376C0000, 0x376D0000, 0x376E0000,
01127		0x37700000, 0x37710000, 0x37720000, 0x37730000, 0x37740000, 0x37750000, 0x37760000, 0x37780000, 0x37790000, 0x37780000, 0x377E0000, 0x377E0000,
01128	0x37838000,	0x37800000, 0x37808000, 0x37810000, 0x37818000, 0x37820000, 0x37828000, 0x37830000, 0x37840000, 0x37848000, 0x37850000, 0x37858000, 0x37868000, 0x37868000, 0x37870000,
01129		0x37880000, 0x37888000, 0x37890000, 0x37898000, 0x378A0000, 0x378A8000, 0x378B0000, 0x378C0000, 0x378C8000, 0x378D0000, 0x378D0000, 0x378E8000, 0x378F0000,
01130		0x37900000, 0x37908000, 0x37910000, 0x37918000, 0x37920000, 0x37928000, 0x37930000, 0x37940000, 0x37948000, 0x37950000, 0x37958000, 0x37960000, 0x37968000, 0x37970000,
01131		0x37980000, 0x37988000, 0x37990000, 0x37998000, 0x379A0000, 0x379A8000, 0x379B0000, 0x379C0000, 0x379C8000, 0x379D0000, 0x379D0000, 0x379E0000, 0x379E0000, 0x379F0000,
01132	0x379F8000, 0x37A38000,	0x37A00000, 0x37A08000, 0x37A10000, 0x37A18000, 0x37A20000, 0x37A28000, 0x37A30000, 0x37A440000, 0x37A48000, 0x37A50000, 0x37A50000, 0x37A60000, 0x37A68000, 0x37A70000,
01133	0x37A78000, 0x37AB8000,	0x37A80000, 0x37A88000, 0x37A90000, 0x37A98000, 0x37AA0000, 0x37AA8000, 0x37AB0000, 0x37AC0000, 0x37AC8000, 0x37AD0000, 0x37AD0000, 0x37AE0000, 0x37AE0000,
01134	0x37AF8000, 0x37B38000,	0x37B00000, 0x37B08000, 0x37B10000, 0x37B18000, 0x37B20000, 0x37B28000, 0x37B30000, 0x37B48000, 0x37B50000, 0x37B50000, 0x37B60000, 0x37B60000, 0x37B70000,
01135	0x37B78000, 0x37BB8000,	0x37B80000, 0x37B88000, 0x37B90000, 0x37B98000, 0x37BA0000, 0x37BA8000, 0x37BB0000, 0x37BC8000, 0x37BC8000, 0x37BD0000, 0x37BC8000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x
01136	0x37BF8000,	0x37C00000, 0x37C08000, 0x37C10000, 0x37C18000, 0x37C20000, 0x37C28000, 0x37C30000, 0x37C40000, 0x37C48000, 0x37C50000, 0x37C50000, 0x37C60000, 0x37C68000, 0x37C70000,
01137	0x37C78000,	0x37C80000, 0x37C88000, 0x37C90000, 0x37C98000, 0x37CA0000, 0x37CA8000, 0x37CB0000, 0x37CC0000, 0x37CC8000, 0x37CD0000, 0x37CD0000, 0x37CE0000, 0x37CE0000, 0x37CF0000,
01138	0x37CF8000,	0x37D00000, 0x37D48000, 0x37D50000, 0x37D58000, 0x37D60000, 0x37D68000, 0x37D700000,
01139	0x37D78000,	0x37D80000, 0x37D80000, 0x37D90000, 0x37D90000, 0x37D80000, 0x37D800000, 0x37D800000, 0x37D800000, 0x37D800000, 0x37D8000000, 0x37D80000000, 0x37D8000000, 0x37D8000000000, 0x37D800000000, 0x37D80000000, 0x37D80000000000, 0x37D800000000, 0x37D8000000000000000000000000000000000000
01140	0x37DF8000,	0x37E00000, 0x37E08000, 0x37E10000, 0x37E18000, 0x37E20000, 0x37E28000, 0x37E30000,
01141	0x37E78000,	0x37E40000, 0x37E48000, 0x37E50000, 0x37E58000, 0x37E60000, 0x37E68000, 0x37E70000, 0x37E80000, 0x37E
01142	0x37EB8000, 0x37EF8000,	0x37EC0000, 0x37EC8000, 0x37ED0000, 0x37ED8000, 0x37EE8000, 0x37EE8000, 0x37EF0000, 0x37F00000, 0x37F10000, 0x37F1
01143	0x37F38000, 0x37F78000,	0x37F40000, 0x37F48000, 0x37F50000, 0x37F58000, 0x37F60000, 0x37F68000, 0x37F70000, 0x37F80000, 0x37F80000, 0x37F80000, 0x37F80000, 0x37F90000, 0x37F90000, 0x37F90000, 0x37F90000, 0x37F90000, 0x37F90000, 0x37F90000, 0x37F
01144	0x37FB8000, 0x37FF8000,	0x37FC0000, 0x37FC8000, 0x37FD0000, 0x37FD8000, 0x37FE0000, 0x37FE8000, 0x37FF0000, 0x38000000, 0x38004000, 0x38008000, 0x38000000, 0x38014000, 0x38018000,
01145	0x3801C000, 0x3803C000,	0x38020000, 0x38024000, 0x38028000, 0x3802c000, 0x38030000, 0x38034000, 0x38038000, 0x38040000, 0x38044000, 0x38048000, 0x3804c000, 0x38050000, 0x38050000, 0x38050000,
01146	0x3805C000, 0x3807C000,	0x38060000, 0x38064000, 0x38068000, 0x3806C000, 0x38070000, 0x38074000, 0x38078000, 0x38080000, 0x38084000, 0x38088000, 0x3808C000, 0x38090000, 0x38094000, 0x38098000,
01147	0x3809C000, 0x380BC000,	0x380A0000, 0x380A4000, 0x380A8000, 0x380AC000, 0x380B0000, 0x380B4000, 0x380B8000, 0x380C0000, 0x380C4000, 0x380C8000, 0x380C0000, 0x380D8000, 0x380C0000, 0x380D8000,
01117	0x380DC000, 0x380FC000,	0x380E0000, 0x380E4000, 0x380E8000, 0x380EC000, 0x380F0000, 0x380F4000, 0x380F8000, 0x38100000, 0x38104000, 0x38108000, 0x38100000, 0x38114000, 0x38118000,
	0x3811C000, 0x3813C000,	0x38120000, 0x38124000, 0x38128000, 0x38120000, 0x38130000, 0x38134000, 0x38138000,
01149	0x3815C000, 0x3817C000,	0x38140000, 0x38144000, 0x38148000, 0x38140000, 0x38150000, 0x38154000, 0x38158000, 0x38160000, 0x38164000, 0x38168000, 0x38160000, 0x38174000, 0x38174000, 0x38178000,
01150	0x3819C000, 0x381BC000,	0x38180000, 0x38184000, 0x38188000, 0x3818C000, 0x38190000, 0x38194000, 0x38198000, 0x381A0000, 0x30000, 0x30000, 0x300000, 0x300000, 0x30000, 0x300000, 0x300000, 0x300000, 0x30000000, 0
01151	0x381DC000, 0x381FC000,	0x381C0000, 0x381C4000, 0x381C8000, 0x381CC000, 0x381D0000, 0x381D4000, 0x381D8000, 0x381E0000, 0x381E4000, 0x381E8000, 0x381E0000, 0x381F4000, 0x381F8000,
01152	0x3821C000, 0x3823C000,	0x38200000, 0x38204000, 0x38208000, 0x3820C000, 0x38210000, 0x38214000, 0x38218000, 0x38220000, 0x38224000, 0x38228000, 0x3822C000, 0x38230000, 0x38234000, 0x38238000,
01153	0x3825C000,	0x38240000, 0x38244000, 0x38248000, 0x3824C000, 0x38250000, 0x38254000, 0x38258000, 0x38260000, 0x38264000, 0x38268000, 0x3826C000, 0x38270000, 0x38274000, 0x38278000,

	0x3827C000,	
01154	ŕ	0x38280000, 0x38284000, 0x38288000, 0x3828C000, 0x38290000, 0x38294000, 0x38298000, 0x382A0000, 0x382A4000, 0x382A8000, 0x382AC000, 0x382B0000, 0x382B4000, 0x382B8000,
01155		0x382C0000, 0x382C4000, 0x382C8000, 0x382CC000, 0x382D0000, 0x382D4000, 0x382E4000, 0x382E4000, 0x382E8000, 0x382E0000, 0x382F4000, 0x382F8000,
01156		0x38300000, 0x38304000, 0x38308000, 0x3830C000, 0x38310000, 0x38314000, 0x38318000, 0x38320000, 0x38324000, 0x38328000, 0x3832C000, 0x38330000, 0x38334000, 0x38338000,
01157		0x38340000, 0x38344000, 0x38348000, 0x3834C000, 0x38350000, 0x38354000, 0x38358000, 0x38360000, 0x38364000, 0x38368000, 0x3836C000, 0x38370000, 0x38374000, 0x38378000,
01158	0x3839C000, 0x383BC000,	0x38380000, 0x38384000, 0x38388000, 0x3838C000, 0x38390000, 0x38394000, 0x38398000, 0x383A0000, 0x383A4000, 0x383A8000, 0x383AC000, 0x383B0000, 0x383B4000, 0x383B8000,
01159	0x383DC000, 0x383FC000,	0x383C0000, 0x383C4000, 0x383C8000, 0x383CC000, 0x383D0000, 0x383D4000, 0x383D8000, 0x383E4000, 0x383E4000, 0x383E4000, 0x383E4000, 0x383F4000, 0x383F4000, 0x383F8000,
01160	0x3841C000, 0x3843C000,	0x38400000, 0x38404000, 0x38408000, 0x3840C000, 0x38410000, 0x38414000, 0x38418000, 0x38420000, 0x38424000, 0x38428000, 0x38428000, 0x38428000, 0x38438000, 0x38438000,
01161	0x3845C000, 0x3847C000,	0x38440000, 0x38444000, 0x38448000, 0x3844C000, 0x38450000, 0x38454000, 0x38458000, 0x38460000, 0x38464000, 0x38468000, 0x38460000, 0x38470000, 0x38474000, 0x38478000, 0x38480000, 0x38480000, 0x38480000, 0x38480000, 0x38480000, 0x38498000, 0x3849000, 0x38490000, 0x38490
01163	0x3849C000, 0x384BC000,	0x384A0000, 0x384A4000, 0x384A8000, 0x384AC000, 0x384B0000, 0x384B4000, 0x384B8000, 0x384C0000, 0x384C4000, 0x384C000, 0x384D0000, 0x30000, 0x30000, 0x3000000, 0x3000000, 0x300000000, 0x30000000, 0x30000000, 0x30000000000
01164	0x384DC000, 0x384FC000,	0x384E0000, 0x384E4000, 0x384E8000, 0x384E0000, 0x384F0000, 0x384F4000, 0x384F8000, 0x3850000, 0x38504000, 0x38500000, 0x3850000, 0x38500000, 0x38500000000000000000000000000000000000
01165	0x3851C000, 0x3853C000,	0x38520000, 0x38524000, 0x38528000, 0x38520000, 0x38530000, 0x38534000, 0x38538000, 0x38540000, 0x38544000, 0x38544000, 0x38540000, 0x38558000,
01166	0x3855C000, 0x3857C000,	0x38560000, 0x38564000, 0x38568000, 0x3856C000, 0x38570000, 0x38574000, 0x38578000, 0x38580000, 0x38584000, 0x38588000, 0x38580000, 0x38598000, 0x38598000,
01167	0x385BC000,	0x385A0000, 0x385A4000, 0x385A8000, 0x385AC000, 0x385B0000, 0x385B4000, 0x385B8000, 0x385C0000, 0x385C4000, 0x385C8000, 0x385C0000, 0x385D4000, 0x385D8000,
01168	0x385FC000,	0x385E0000, 0x385E4000, 0x385E8000, 0x385E0000, 0x385F0000, 0x385F4000, 0x385F8000, 0x38600000, 0x38604000, 0x38608000, 0x38600000, 0x38614000, 0x38618000,
01169	0x3863C000,	0x38620000, 0x38624000, 0x38628000, 0x38620000, 0x38630000, 0x38634000, 0x38638000, 0x38640000, 0x38644000, 0x38648000, 0x3864000, 0x38650000, 0x38654000, 0x38658000,
01170	0x3867C000,	0x38660000, 0x38664000, 0x38668000, 0x3866C000, 0x38670000, 0x38674000, 0x38678000, 0x38680000, 0x38684000, 0x38688000, 0x3868C000, 0x38690000, 0x38694000, 0x38698000, 0x386A0000, 0x386A4000, 0x386A8000, 0x386AC000, 0x386B0000, 0x386B4000, 0x386B8000,
01171	0x386BC000,	0x386C0000, 0x386C4000, 0x386E8000, 0x386C0000, 0x386D0000, 0x386D00000, 0x386D00000, 0x386D00000, 0x3000000, 0x3000000, 0x3000000, 0x3000000, 0x30000000, 0x300000000, 0x30000000000
01172	0x386FC000,	0x38700000, 0x38704000, 0x38708000, 0x38700000, 0x38710000, 0x38714000, 0x38718000, 0x38720000, 0x38724000, 0x38728000, 0x38720000, 0x38734000, 0x38738000,
01173		0x38740000, 0x38744000, 0x38748000, 0x38740000, 0x38750000, 0x38754000, 0x38758000, 0x38760000, 0x38764000, 0x38768000, 0x38770000, 0x38774000, 0x38778000,
01174		0x38780000, 0x38784000, 0x38788000, 0x3878C000, 0x38790000, 0x38794000, 0x38798000, 0x387A0000, 0x387A4000, 0x387A8000, 0x387AC000, 0x387B0000, 0x387B4000, 0x387B8000,
01175		0x387C0000, 0x387C4000, 0x387C8000, 0x387CC000, 0x387D0000, 0x387D4000, 0x387D8000, 0x387E0000, 0x387E4000, 0x387E8000, 0x387E0000, 0x387F4000, 0x387F8000,
01176	0x387FC000, 0x3800E000, 0x3801E000,	0x38000000, 0x38002000, 0x38004000, 0x38006000, 0x38008000, 0x3800A000, 0x3800C000, 0x38010000, 0x38012000, 0x38014000, 0x38016000, 0x38018000, 0x3801A000, 0x3801C000,
01177	ŕ	0x38020000, 0x38022000, 0x38024000, 0x38026000, 0x38028000, 0x3802A000, 0x3802C000, 0x38030000, 0x38032000, 0x38034000, 0x38036000, 0x38038000, 0x3803A000, 0x3803C000,
01178		0x38040000, 0x38042000, 0x38044000, 0x38046000, 0x38048000, 0x3804A000, 0x3804C000, 0x38050000, 0x38052000, 0x38054000, 0x38056000, 0x38058000, 0x3805A000, 0x3805C000,
01179		0x38060000, 0x38062000, 0x38064000, 0x38066000, 0x38068000, 0x3806A000, 0x3806C000, 0x38070000, 0x38072000, 0x38074000, 0x38076000, 0x38078000, 0x3807A000, 0x3807C000,
01180	0x3808E000, 0x3809E000,	0x38080000, 0x38082000, 0x38084000, 0x38086000, 0x38088000, 0x38088000, 0x38080000, 0x38090000, 0x38092000, 0x38094000, 0x38096000, 0x38098000, 0x3809A000, 0x3809C000,
01181	0x380AE000, 0x380BE000,	0x380A0000, 0x380A2000, 0x380A4000, 0x380A6000, 0x380A8000, 0x380A000, 0x380A000, 0x380B0000, 0x380B2000, 0x380B4000, 0x380B6000, 0x380B8000, 0x380BA000, 0x380BC000,
01182	0x380CE000,	0x380C0000, 0x380C2000, 0x380C4000, 0x380C6000, 0x380C8000, 0x380CA000, 0x380C000, 0x380D0000, 0x380D2000, 0x380D4000, 0x380D6000, 0x380D8000, 0x380DA000, 0x380DC000,

	0x380DE000,	
01183	0x380EE000, 0x380FE000,	0x380E0000, 0x380E2000, 0x380E4000, 0x380E6000, 0x380E8000, 0x380EA000, 0x380EC000, 0x380F0000, 0x380F2000, 0x380F4000, 0x380F6000, 0x380F8000, 0x380FA000, 0x380FC000,
01184		0x38100000, 0x38102000, 0x38104000, 0x38106000, 0x38108000, 0x3810A000, 0x3810C000, 0x38110000, 0x38112000, 0x38114000, 0x38116000, 0x38118000, 0x3811A000, 0x3811C000,
01185		0x38120000, 0x38122000, 0x38124000, 0x38126000, 0x38128000, 0x3812A000, 0x3812C000, 0x38130000, 0x38132000, 0x38134000, 0x38136000, 0x38138000, 0x3813A000, 0x3813C000,
01186		0x38140000, 0x38142000, 0x38144000, 0x38146000, 0x38148000, 0x38144000, 0x3814000, 0x38150000, 0x38152000, 0x38154000, 0x38156000, 0x38158000, 0x38154000, 0x38156000,
01187	ŕ	0x38160000, 0x38162000, 0x38164000, 0x38166000, 0x38168000, 0x3816A000, 0x3816C000, 0x38170000, 0x38172000, 0x38174000, 0x38176000, 0x38178000, 0x3817A000, 0x3817C000,
01188	·	0x38180000, 0x38182000, 0x38184000, 0x38186000, 0x38188000, 0x3818A000, 0x3818C000, 0x38190000, 0x38192000, 0x38194000, 0x38196000, 0x38198000, 0x3819A000, 0x3819C000,
01189	0x381AE000, 0x381BE000,	0x381A0000, 0x381A2000, 0x381A4000, 0x381A6000, 0x381A8000, 0x381AA000, 0x381AC000, 0x381B0000, 0x381B2000, 0x381B4000, 0x381B6000, 0x381B8000, 0x381BA000, 0x381BC000,
01190	0x381CE000, 0x381DE000,	0x381C0000, 0x381C2000, 0x381C4000, 0x381C6000, 0x381C8000, 0x381CA000, 0x381CC000, 0x381D0000, 0x381D2000, 0x381D4000, 0x381D6000, 0x381D8000, 0x381DA000, 0x381DC000,
01191	0x381EE000, 0x381FE000,	0x381E0000, 0x381E2000, 0x381E4000, 0x381E6000, 0x381E8000, 0x381EA000, 0x381EC000, 0x381F0000, 0x381F2000, 0x381F4000, 0x381F6000, 0x381F8000, 0x381FA000, 0x381FC000,
01192	0x3820E000, 0x3821E000,	0x38200000, 0x38202000, 0x38204000, 0x38206000, 0x38208000, 0x3820A000, 0x3820C000, 0x38210000, 0x38212000, 0x38214000, 0x38216000, 0x38218000, 0x3821A000, 0x3821C000,
01193	0x3822E000, 0x3823E000,	
01194	0x3824E000, 0x3825E000,	
01195	0x3826E000, 0x3827E000,	
01196	0x3828E000, 0x3829E000,	
01197	0x382AE000, 0x382BE000,	
01198	0x382CE000, 0x382DE000,	
01199	0x382EE000, 0x382FE000,	0x382E0000, 0x382E2000, 0x382E4000, 0x382E6000, 0x382E8000, 0x382EA000, 0x382EC000, 0x382F0000, 0x382F2000, 0x382F4000, 0x382F4000, 0x382F4000, 0x382F4000, 0x382F4000, 0x382F4000, 0x382F4000, 0x38300000, 0x38302000, 0x38304000, 0x38306000, 0x3826000, 0x3826000, 0x38306000, 0x38306000, 0x38306000, 0x38306000, 0x38306000, 0x38306000, 0x38306000, 0x38306000, 0x3826000, 0x3
01200	0x3830E000, 0x3831E000,	0x38310000, 0x38312000, 0x38314000, 0x38316000, 0x38318000, 0x3831A000, 0x38312000, 0x38312000, 0x38312000, 0x38312000, 0x38322000, 0x38322000, 0x38322000, 0x38322000, 0x38322000, 0x38322000, 0x38322000, 0x38322000, 0x38324000, 0x3834000, 0x38324000, 0x3824000, 0x3824000, 0x3824000, 0x3824000, 0x3824000, 0x3824000, 0x3824000, 0x3824000, 0x3824000, 0x38
01201	0x3833E000,	0x38330000, 0x38332000, 0x38334000, 0x38336000, 0x38338000, 0x3833A000, 0x3833C000,
01202		0x38350000, 0x38352000, 0x38354000, 0x38356000, 0x38358000, 0x3835A000, 0x3835C000,
01204	0x3836E000, 0x3837E000,	0x38370000, 0x38372000, 0x38374000, 0x38376000, 0x38378000, 0x3837A000, 0x3837C000,
01205	0x3838E000, 0x3839E000,	0x38390000, 0x38392000, 0x38394000, 0x38396000, 0x38398000, 0x3839A000, 0x3839C000, 0x383A0000, 0x383A2000, 0x383A000, 0x383A00, 0x38A00, 0x38A00, 0x300, 0x300
01206	0x383BE000,	0x383B0000, 0x383B2000, 0x383B4000, 0x383B6000, 0x383B8000, 0x383BA000, 0x383BC000, 0x383C0000, 0x383C2000, 0x383C000, 0x30C00, 0x30C00, 0x30C00, 0x30C00, 0x30C00, 0x30C00, 0x30C00, 0x30C00, 0x30C00, 0x30C0
01207	0x383DE000,	0x383E0000, 0x383E2000, 0x383E4000, 0x383E6000, 0x383E8000, 0x383EA000, 0x383EC000,
01208	0x383FE000,	0x38400000, 0x38402000, 0x38404000, 0x38406000, 0x38408000, 0x3840A000, 0x3840C000,
01209	0x3841E000,	0x38420000, 0x38422000, 0x38424000, 0x38426000, 0x38428000, 0x3842A000, 0x3842C000,
01210	0x3843E000,	0x38440000, 0x38442000, 0x38444000, 0x38446000, 0x38448000, 0x3844A000, 0x3844C000,
01211	0x3845E000,	0x38450000, 0x38452000, 0x38454000, 0x38456000, 0x38458000, 0x3845A000, 0x3845C000, 0x38460000, 0x38462000, 0x38462000, 0x38460000, 0x38462000, 0x38460000, 0x3840000, 0x38400000, 0x3840000, 0x38400000, 0x3840000, 0x38400000, 0x38400000, 0x38400000, 0x38400000, 0x3840000
V1211		0x38470000, 0x38472000, 0x38474000, 0x38476000, 0x38478000, 0x3847A000, 0x3847C000,

	0x3847E000,	
01212	0x3848E000, 0x3849E000,	0x38480000, 0x38482000, 0x38484000, 0x38486000, 0x38488000, 0x3848A000, 0x3848C000, 0x38490000, 0x38492000, 0x38494000, 0x38496000, 0x38498000, 0x3849A000, 0x3849C000,
01213	ŕ	0x384A0000, 0x384A2000, 0x384A4000, 0x384A6000, 0x384A8000, 0x384AA000, 0x384AC000, 0x384B0000, 0x384B2000, 0x384B4000, 0x384B6000, 0x384B0000, 0x384B0000, 0x384B0000,
01214		0x384C0000, 0x384C2000, 0x384C4000, 0x384C6000, 0x384C8000, 0x384CA000, 0x384CC000, 0x384D0000, 0x384D2000, 0x384D4000, 0x384D6000, 0x384D8000, 0x384DA000, 0x384DC000,
01215		0x384E0000, 0x384E2000, 0x384E4000, 0x384E6000, 0x384E8000, 0x384EA000, 0x384EC000, 0x384F0000, 0x384F2000, 0x384F4000, 0x384F6000, 0x384F8000, 0x384FA000, 0x384FC000,
01216		0x38500000, 0x38502000, 0x38504000, 0x38506000, 0x38508000, 0x3850A000, 0x3850C000, 0x38510000, 0x38512000, 0x38514000, 0x38516000, 0x38518000, 0x3851A000, 0x3851C000,
01217		0x38520000, 0x38522000, 0x38524000, 0x38526000, 0x38528000, 0x38522000, 0x38530000, 0x38532000, 0x38534000, 0x38536000, 0x38538000, 0x3853A000, 0x3853C000,
01218		0x38540000, 0x38542000, 0x38544000, 0x38546000, 0x38548000, 0x3854000, 0x38550000, 0x38552000, 0x38554000, 0x38556000, 0x38550000, 0x38550000, 0x38550000,
01219		0x38560000, 0x38562000, 0x38564000, 0x38566000, 0x38568000, 0x3856000, 0x38570000, 0x38572000, 0x38574000, 0x38576000, 0x38578000, 0x38572000, 0x38572000,
01220	0x3858E000, 0x3859E000,	0x38580000, 0x38582000, 0x38584000, 0x38586000, 0x38588000, 0x38588000, 0x3858000, 0x38590000, 0x38592000, 0x38594000, 0x38596000, 0x38598000, 0x3859A000, 0x3859C000,
01221	0x385AE000, 0x385BE000,	0x385A0000, 0x385A2000, 0x385A4000, 0x385A6000, 0x385A8000, 0x385A0000, 0x385A0000, 0x385B0000, 0x385B2000, 0x385B4000, 0x385B6000, 0x385B8000, 0x385B0000, 0x385B
01222	0x385CE000, 0x385DE000,	0x385C0000, 0x385C2000, 0x385C4000, 0x385C6000, 0x385C8000, 0x385CA000, 0x385CC000, 0x385D0000, 0x385D2000, 0x385D4000, 0x385D6000, 0x385D8000, 0x385DA000, 0x385DC000,
01223	0x385EE000, 0x385FE000,	0x385E0000, 0x385E2000, 0x385E4000, 0x385E6000, 0x385E8000, 0x385EA000, 0x385EC000, 0x385F0000, 0x385F2000, 0x385F4000, 0x385F6000, 0x385F8000, 0x385FA000, 0x385FC000,
01224	0x3860E000, 0x3861E000,	0x38600000, 0x38602000, 0x38604000, 0x38606000, 0x38608000, 0x3860A000, 0x3860C000, 0x38610000, 0x38612000, 0x38614000, 0x38616000, 0x38618000, 0x3861A000, 0x3861C000,
01225	0x3862E000, 0x3863E000,	0x38620000, 0x38622000, 0x38624000, 0x38626000, 0x38628000, 0x3862A000, 0x3862C000, 0x38630000, 0x38632000, 0x38634000, 0x38636000, 0x3863A000, 0x3863C000,
01226	0x3864E000, 0x3865E000,	0x38640000, 0x38642000, 0x38644000, 0x38646000, 0x38648000, 0x3864A000, 0x3864C000, 0x38650000, 0x38652000, 0x38654000, 0x38656000, 0x38658000, 0x3865A000, 0x3865C000,
01227	0x3866E000, 0x3867E000,	0x38660000, 0x38662000, 0x38664000, 0x38666000, 0x38668000, 0x3866A000, 0x3866C000, 0x38670000, 0x38672000, 0x38674000, 0x38676000, 0x38678000, 0x3867A000, 0x3867C000,
01228	0x3868E000, 0x3869E000,	0x38680000, 0x38682000, 0x38684000, 0x38686000, 0x38688000, 0x3868A000, 0x3868C000, 0x38690000, 0x38692000, 0x38694000, 0x38696000, 0x3869A000, 0x3869C000,
01229	0x386AE000, 0x386BE000,	0x386A0000, 0x386A2000, 0x386A4000, 0x386A6000, 0x386A8000, 0x386AA000, 0x386AC000, 0x386B0000, 0x386B2000, 0x386B4000, 0x386B6000, 0x386B8000, 0x386BA000, 0x386BC000,
01230	0x386DE000,	0x386C0000, 0x386C2000, 0x386C4000, 0x386C6000, 0x386C8000, 0x386CA000, 0x386CC000, 0x386D0000, 0x386D2000, 0x386D4000, 0x386D6000, 0x386D8000, 0x386DA000, 0x386DC000,
01231		0x386E0000, 0x386E2000, 0x386E4000, 0x386E6000, 0x386E8000, 0x386EA000, 0x386EC000, 0x386F0000, 0x386F2000, 0x386F4000, 0x386F
01232	0x3870E000, 0x3871E000,	0x38700000, 0x38702000, 0x38704000, 0x38706000, 0x38708000, 0x3870A000, 0x3870C000, 0x38710000, 0x38712000, 0x38714000, 0x38716000, 0x38718000, 0x3871A000, 0x3870A000, 0x3870
01233	0x3872E000, 0x3873E000,	0x38720000, 0x38722000, 0x38724000, 0x38726000, 0x38728000, 0x3872A000, 0x3872C000, 0x38730000, 0x38732000, 0x38734000, 0x38734000, 0x38736000, 0x3873A000, 0x3873C000, 0x38740000, 0x38742000, 0x38740000, 0x38742000, 0x38740000, 0x3874
01234	0x3874E000, 0x3875E000,	0x38740000, 0x38742000, 0x38744000, 0x38746000, 0x38746000, 0x38750000, 0x38750000, 0x38750000, 0x38750000, 0x38760000, 0x3870000, 0x3870000, 0x38700000, 0x38700000, 0x38700000, 0x387000
01235	0x3876E000, 0x3877E000,	0x38770000, 0x38772000, 0x38774000, 0x38776000, 0x38778000, 0x3877A000, 0x3877A000, 0x3877C000, 0x38780000, 0x38782000, 0x38784000, 0x38786000, 0x38788000, 0x3878000, 0x3878000,
01236	0x3878E000, 0x3879E000,	0x38780000, 0x38782000, 0x38784000, 0x38786000, 0x38786000, 0x38786000, 0x38796000, 0x38796000, 0x38796000, 0x38796000, 0x38786000, 0x3878000, 0x3879000, 0x3879
01237	0x387AE000, 0x387BE000,	0x387A0000, 0x387A2000, 0x387A4000, 0x387A6000, 0x387A6000, 0x387B0000, 0x387B0000, 0x387B0000, 0x387B0000, 0x387B0000, 0x387C0000, 0x30000, 0x30000, 0x30000, 0x30000, 0x0000, 0x00000, 0x00000, 0x00000, 0x00000, 0x00000, 0x000000, 0x0000000, 0x00000000
01238	0x387CE000, 0x387DE000,	0x387D0000, 0x387D2000, 0x387D4000, 0x387D6000, 0x387D8000, 0x387DA000, 0x387DC000, 0x387D0000, 0x387E2000, 0x387E4000, 0x387E6000, 0x387E8000, 0x387E2000, 0x387E2000,
01239	0x387EE000, 0x387FE000	0x387F0000, 0x387F2000, 0x387F4000, 0x387F6000, 0x387F8000, 0x387FA000, 0x387FC000,
01240		0x00000000, 0x00800000, 0x01000000, 0x01800000, 0x02000000, 0x02800000, 0x03000000,

```
0x03800000, 0x04000000, 0x04800000, 0x05000000, 0x05800000, 0x06000000, 0x06800000, 0x07000000,
01242
                                   0x08000000, 0x08800000, 0x09000000, 0x09800000, 0x0A000000, 0x0A800000, 0x0B000000,
          0x0B800000, 0x0C000000, 0x0C800000, 0x0D0000000, 0x0B800000, 0x0E000000, 0x0E800000, 0x0F000000,
          0 \times 47800000.
                                   0x80000000, 0x80800000, 0x81000000, 0x81800000, 0x82000000, 0x82800000, 0x83000000,
01243
          0x83800000,\ 0x84000000,\ 0x84800000,\ 0x885000000,\ 0x85800000,\ 0x86000000,\ 0x86800000,\ 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] = {
         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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 102
01246
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, 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;
                       #endif
01253
01254
01255
01259
                       inline double half2float_impl(unsigned int value, double, true_type)
01260
01261
                      #if HALF ENABLE F16C INTRINSICS
                            return _mm_cvtsd_f64(_mm_cvtps_pd(_mm_cvtph_ps(_mm_cvtsi32_si128(value))));
01262
01263
                       #else
01264
                            uint32 hi = static_cast<uint32>(value&0x8000) « 16;
01265
                             unsigned int abs = value & 0x7FFF;
01266
                             if(abs)
01267
                                   hi |= 0x3F000000 « static_cast<unsigned>(abs>=0x7C00);
01268
                                   for(; abs<0x400; abs<=1,hi-=0x100000);</pre>
01269
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)
                                   out = (std::numeric_limits<T>::has_signaling_NaN && !(abs&0x200)) ?
01288
        std::numeric_limits<T>::signaling_NaN() :
01289
                                         std::numeric_limits<T>::has_quiet_NaN ? std::numeric_limits<T>::quiet_NaN() : T();
                            else if(abs == 0x7C00)
01290
01291
                                  out = std::numeric limits<T>::has infinity ? std::numeric limits<T>::infinity() :
         std::numeric_limits<T>::max();
01292
                            else if (abs > 0x3FF)
01293
                                   out = std::ldexp(static_cast<T>((abs&0x3FF)|0x400), (abs>10)-25);
                             else
01294
01295
                                  out = std::ldexp(static cast<T>(abs), -24);
01296
                            return (value&0x8000) ? -out : out;
01297
                      }
01298
01303
                       template<typename T> T half2float(unsigned int value)
01304
                      {
01305
                             return half2float_impl(value, T(),
         bool_type<std::numeric_limits<T>::is_iec559&&sizeof(typename bits<T>::type) == sizeof(T)>());
01306
                      }
01307
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;
                             if(abs >= 0x7C00)
01320
01321
01322
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);
                                   return (R==std::round_toward_infinity) ? T(~(value»15)&(abs!=0)) :
01328
                                                 (R==std::round_toward_neg_infinity) ? -T(value>0x8000) :
01329
01330
                                                T();
01331
                             int exp = 25 - (abs * 10);
01332
                             unsigned int m = (value \& 0x3FF) | 0x400;
01333
```

```
int32 i = static\_cast < int32 > ((exp <= 0) ? (m <-exp) : ((m + (
                         (R==std::round\_to\_nearest) ? ((1 \ll (exp-1)) - (\sim (m \gg exp) \& E)) :
01335
01336
                          (R = std::round\_toward\_infinity) ? (((1 \ll exp)-1) & ((value \gg 15)-1)) :
                         (R==std::round_toward_neg_infinity) ? (((1«exp)-1)&-(value»15)) : 0)) »exp));
01337
      if((!std::numeric_limits<T>::is_signed && (value&0x8000)) ||
(std::numeric_limits<T>::digits<16 &&</pre>
01338
                         ((value&0x8000) ? (-i<std::numeric_limits<T>::min()) :
      (i>std::numeric_limits<T>::max()))))
01340
                        raise(FE_INVALID);
                    else if(I && exp > 0 && (m&((1«exp)-1)))
    raise(FE_INEXACT);
01341
01342
                    return static cast<T>((value&0x8000) ? -i : i);
01343
01344
                }
01345
01349
01355
                template<std::float_round_style R> uint32 mulhi(uint32 x, uint32 y)
01356
                    uint32 xy = (x \approx 16) * (y \approx 0 \times FFFF), yx = (x \approx 0 \times FFFF) * (y \approx 16), c = (x \approx 0 \times FFFF) + (y \approx 0 \times FFFF) +
01357
       (((x&0xFFFF) * (y&0xFFFF)) »16);
               return (x»16) * (y»16) + (xy»16) + (yx»16) + (c»16) +
01358
                         ((R==std::round_to_nearest) ? ((c»15)&1) : (R==std::round_toward_infinity) ?
01359
       ((c&0xFFFF)!=0) : 0);
01360
               }
01361
                inline uint32 multiply64(uint32 x, uint32 y)
01366
01367
01368
                #if HALF_ENABLE_CPP11_LONG_LONG
01369
                    return static_cast<uint32>((static_cast<unsigned long long>(x)*static_cast<unsigned long</pre>
      long>(y)+0x80000000)»32);
01370
                #else
01371
                    return mulhi<std::round to nearest>(x, v);
01372
                #endif
01373
01374
01380
                inline uint32 divide64(uint32 x, uint32 y, int &s)
01381
01382
                #if HALF ENABLE CPP11 LONG LONG
                    unsigned long long xx = static_cast<unsigned long long>(x) « 32;
01383
01384
                     return s = (xx%y!=0), static_cast<uint32>(xx/y);
01385
01386
                    y »= 1;
                    uint32 rem = x, div = 0;

for(unsigned int i=0; i<32; ++i)
01387
01388
01389
01390
01391
                         if(rem >= y)
01392
01393
                             rem -= y;
01394
                             div \mid = 1;
01395
01396
                         rem «= 1;
01397
01398
                     return s = rem > 1, div;
01399
                #endif
01400
01401
01409
                template<br/>bool Q,bool R> unsigned int mod(unsigned int x, unsigned int y, int *quo = NULL)
01410
01411
                    unsigned int q = 0;
01412
                     if(x > y)
01413
                         int absx = x, absy = y, expx = 0, expy = 0;
for(; absx<0x400; absx<=1,--expx);
for(; absy<0x400; absy<=1,--expy);</pre>
01414
01415
01416
01417
                         expx += absx » 10;
                         expy += absy » 10;
01418
01419
                         int mx = (absx&0x3FF) \mid 0x400, my = (absy&0x3FF) \mid 0x400;
                         for(int d=expx-expy; d; --d)
01420
01421
01422
                              if(!Q && mx == my)
01423
                                  return 0;
01424
                              if(mx >= my)
01425
                              {
01426
                                  mx -= my;
                                  q += Q;
01427
01428
01429
                             mx «= 1;
01430
                              q <= static_cast<int>(Q);
01431
                         if(!0 \&\& mx == mv)
01432
                             return 0;
01433
01434
                         if(mx >= my)
01435
                             mx -= my;
01436
01437
                             ++q;
01438
01439
                         if(Q)
```

```
01440
                        {
01441
                            q &= (1«(std::numeric_limits<int>::digits-1)) - 1;
01442
                             if(!mx)
01443
                                 return *quo = q, 0;
01444
                        for(; mx<0x400; mx«=1,--expy);</pre>
01445
                        x = (expy>0) ? ((expy <10) | (mx & 0x3FF)) : (mx > (1-expy));
01446
01447
                    if(R)
01448
01449
01450
                        unsigned int a, b;
                        if(y < 0x800)
01451
01452
01453
                            a = (x<0x400) ? (x < 1) : (x+0x400);
01454
                            b = y;
01455
01456
                        else
01457
01458
                            a = x;
                            b = y - 0x400;
01459
01460
01461
                        if(a > b | | (a == b \&\& (q&1)))
01462
                            int \exp = (y*10) + (y<0x3FF), d = \exp - (x*10) - (x<0x3FF);
int m = (((y*0x3FF) | ((y>0x3FF) *(10)) *(1) - (((x*0x3FF) | ((x>0x3FF) *(10)) *(1-d));
01463
01464
                            for(; m<0x800 && exp>1; m«=1,--exp);
01465
01466
                            x = 0x8000 + ((exp-1) \times 10) + (m \times 1);
01467
                            q += Q;
01468
                        }
01469
01470
                    if(0)
01471
                        *quo = q;
01472
                    return x;
01473
01474
               template<unsigned int F> uint32 sgrt(uint32 &r, int &exp)
01480
01481
01482
                    int i = exp \& 1;
01483
                    r «= i;
01484
                    exp = (exp-i) / 2;
01485
                    uint32 m = 0:
                    for(uint32 bit=static_cast<uint32>(1) «F; bit; bit»=2)
01486
01487
01488
                        if(r < m+bit)</pre>
01489
                           m \gg = 1;
01490
                        else
01491
                            r -= m + bit;
01492
                            m = (m \gg 1) + bit;
01493
01494
01495
01496
01497
               }
01498
               inline uint32 exp2(uint32 m, unsigned int n = 32)
01504
01505
                    static const uint32 logs[] = {
01507
                        0x80000000, 0x4AE00D1D, 0x2934F098, 0x15C01A3A, 0x0B31FB7D, 0x05AEB4DD, 0x02DCF2D1,
      0x016FE50B,
01508
                        0x00B84E23, 0x005C3E10, 0x002E24CA, 0x001713D6, 0x000B8A47, 0x0005C53B, 0x0002E2A3,
      0x00017153,
01509
                        0x0000B8AA, 0x00005C55, 0x00002E2B, 0x00001715, 0x00000B8B, 0x000005C5, 0x000002E3,
      0x00000171,
                        0x000000B9, 0x0000005C, 0x00000002E, 0x000000017, 0x0000000C, 0x00000006, 0x00000003,
      0x00000001 };
01511
                   if(!m)
                   return 0x80000000;
uint32 mx = 0x80000000, my = 0;
01512
01513
01514
                    for (unsigned int i=1; i<n; ++i)</pre>
01515
01516
                        uint32 mz = my + logs[i];
01517
                        if(mz \ll m)
01518
                        {
01519
                            my = mz;
01520
                            mx += mx » i;
01521
01522
01523
                    return mx;
01524
01525
               inline uint32 log2(uint32 m, unsigned int n = 32)
01531
01532
               {
01533
                    static const uint32 logs[] = -
                        0x80000000, 0x4AE00D1D, 0x2934F098, 0x15C01A3A, 0x0B31FB7D, 0x05AEB4DD, 0x02DCF2D1,
01534
      0x016FE50B,
01535
                        0x00B84E23, 0x005C3E10, 0x002E24CA, 0x001713D6, 0x000B8A47, 0x0005C53B, 0x0002E2A3,
      0x00017153.
```

```
01536
                        0x0000B8AA, 0x00005C55, 0x000002E2B, 0x00001715, 0x00000B8B, 0x000005C5, 0x000002E3,
      0x00000171,
01537
                       0x000000B9, 0x0000005C, 0x0000002E, 0x00000017, 0x0000000C, 0x00000006, 0x00000003,
      0x00000001 };
                   if(m == 0x40000000)
01538
01539
                       return 0:
                   uint32 mx = 0x40000000, my = 0;
01540
                   for(unsigned int i=1; i<n; ++i)</pre>
01541
01542
01543
                       uint32 mz = mx + (mx » i);
01544
                       if(mz \le m)
01545
                       {
01546
                           mx = mz;
                           my += logs[i];
01547
01548
01549
01550
                   return my;
              }
01551
01552
               inline std::pair<uint32, uint32> sincos(uint32 mz, unsigned int n = 31)
01559
01560
                   static const uint32 angles[] = {
                       0x3243F6A9, 0x1DAC6705, 0x0FADBAFD, 0x07F56EA7, 0x03FEAB77, 0x01FFD55C, 0x00FFFAAB,
01561
      0x007FFF55.
01562
                       0x003FFFEB, 0x001FFFFD, 0x00100000, 0x00080000, 0x00040000, 0x00020000, 0x00010000,
      0x00008000,
01563
                       0x00000080,
01564
                       0x00000040, 0x00000020, 0x00000010, 0x000000008, 0x00000004, 0x00000002, 0x00000001 };
                   uint32 mx = 0x26DD3B6A, my = 0;
01565
01566
                   for (unsigned int i=0; i<n; ++i)
01567
                   {
01568
                        uint32 sign = sign_mask(mz);
                       uint32 tx = mx - (arithmetic_shift(my, i)^sign) + sign;
uint32 ty = my + (arithmetic_shift(mx, i)^sign) - sign;
01569
01570
01571
                       mx = tx; my = ty; mz -= (angles[i]^sign) - sign;
01572
01573
                   return std::make_pair(my, mx);
01574
               }
01575
01582
               inline uint32 atan2 (uint32 my, uint32 mx, unsigned int n = 31)
01583
               {
                   static const uint32 angles[] = {
01584
                       0x3243F6A9, 0x1DAC6705, 0x0FADBAFD, 0x07F56EA7, 0x03FEAB77, 0x01FFD55C, 0x00FFFAAB,
01585
      0x007FFF55,
01586
                       0x003FFFEB, 0x001FFFFD, 0x00100000, 0x00080000, 0x00040000, 0x00020000, 0x00010000,
      0x00008000,
01587
                       0 \times 00004000, 0 \times 00002000, 0 \times 00001000, 0 \times 00000800, 0 \times 00000400, 0 \times 00000200, 0 \times 00000100,
      0x00000080,
01588
                       0x00000040, 0x00000020, 0x00000010, 0x000000008, 0x00000004, 0x00000002, 0x00000001 };
01589
                   uint32 mz = 0;
01590
                   for(unsigned int i=0; i<n; ++i)</pre>
01591
01592
                       uint32 sign = sign_mask(my);
                       uint32 tx = mx + (arithmetic_shift(my, i)^sign) - sign;
uint32 ty = my - (arithmetic_shift(mx, i)^sign) + sign;
01593
01594
                       mx = tx; my = ty; mz += (angles[i]^sign) - sign;
01595
01596
01597
                   return mz;
01598
              }
01599
01604
               inline uint32 angle_arg(unsigned int abs, int &k)
01605
                   uint32 m = (abs&0x3FF) | ((abs>0x3FF) <10);
01606
                   int exp = (abs *10) + (abs <= 0x3FF) - 15;
01607
01608
                   if(abs < 0x3A48)
01609
                       return k = 0, m \ll (exp+20);
               #if HALF_ENABLE_CPP11_LONG_LONG
01610
                  unsigned long long y = m * 0xA2F9836E4E442, mask = (1ULL*(62-exp)) - 1, yi = (y+(mask*1))
01611
      & \simmask, f = y - yi;
01612
                   uint32 sign = -static_cast<uint32>(f»63);
01613
                   k = static\_cast < int > (yi» (62-exp));
01614
                   return (multiply64(static_cast<uint32>((sign ? -f : f)»(31-exp)), 0xC90FDAA2)^sign) -
      sign;
01615
              #else
01616
                   uint32 vh = m*0xA2F98 + mulhi < std::round toward zero > (m. 0x36E4E442), vl = (m*0x36E4E442)
      & OxFFFFFFF;
01617
                   uint32 mask = (static_cast<uint32>(1) ((30-exp)) - 1, yi = (yh+(mask»1)) & ~mask, sign =
      -static_cast<uint32>(yi>yh);
                   k = static_cast<int>(yi»(30-exp));
uint32 fh = (yh^sign) + (yi^sign) - ~sign, fl = (yl^sign) - sign;
01618
01619
                   return (multiply64((exp>-1) ? (((fh«(1+exp))&0xFFFFFFFF))(((fl&0xFFFFFFFF))*(31-exp))) : fh,
01620
      0xC90FDAA2) ^sign) - sign;
01621
               #endif
01622
01623
               inline std::pair<uint32, uint32> atan2 args(unsigned int abs)
01627
```

```
{
                  int exp = -15;
01629
01630
                  for(; abs<0x400; abs«=1,--exp);</pre>
01631
                  exp += abs » 10;
                  uint32 my = ((abs\&0x3FF)|0x400) « 5, r = my * my;
01632
                  int rexp = 2 * exp;
01633
                  r = 0x40000000 - ((rexp>-31) ? ((r»-rexp)|((r&((static_cast<uint32>(1) w-rexp)-1))!=0)) :
01634
01635
                   for(rexp=0; r<0x40000000; r«=1,--rexp);</pre>
01636
                  uint32 mx = sqrt<30>(r, rexp);
                  int d = exp - rexp;
01637
                  <u>if</u>(d < 0)
01638
                      return std::make_pair((d<-14) ? ((my*(-d-14))+((my*(-d-15))&1)) : (my*(14+d)),
01639
      (mx \ll 14) + (r \ll 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 \times (14-d)) + (r \times (13-d))/mx));
01642
                  return std::make_pair(my«13, (mx«13)+(r«12)/mx);
01643
01644
              inline std::pair<uint32, uint32> hyperbolic_args(unsigned int abs, int &exp, unsigned int n =
01650
     32)
01651
              {
                  uint32 mx = detail::multiply64(static cast<uint32>((abs&0x3FF)+((abs>0x3FF)x10))x21,
01652
     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 \gg (45-e);
01662
                      mx = (mx \ll (e-14)) \& 0x7FFFFFFF;
01663
                  mx = exp2 (mx, n);
int d = exp « 1, s;
01664
01665
01666
                   if(mx > 0x80000000)
01667
                      my = divide64(0x80000000, mx, s);
01668
01669
                      my \mid = s;
01670
                      ++d:
01671
                   }
01672
                  else
01673
                      my = mx;
01674
                   return std::make_pair(mx, (d<31) ? ((my*d) | ((my*d((static_cast<uint32>(1)*d)-1))!=0)) : 1);
01675
01676
              template<std::float_round_style R> unsigned int exp2_post(uint32 m, int exp, bool esign,
01688
     unsigned int sign = 0, unsigned int n = 32)
01689
01690
                   if(esign)
01691
                       exp = -exp - (m!=0);
01692
                       if(exp < -25)
01693
                          return underflow<R>(sign);
01694
01695
                       else if (exp == -25)
01696
                          return rounded<R, false>(sign, 1, m!=0);
01697
                  else if(exp > 15)
01698
01699
                      return overflow<R>(sign);
01700
                  if(!m)
01701
                      return sign | (((exp+=15)>0) ? (exp«10) : check_underflow(0x200»-exp));
                  m = exp2(m, n);
01702
01703
                   int s = 0;
01704
                  if(esign)
                      m = divide64 (0x80000000, m, s);
01705
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
                  01724
                  <u>if(!m)</u>
01725
                      return 0;
                   for(; m<0x80000000; m«=1,--exp);</pre>
01726
01727
                  int i = m >= L, s;
01728
                  exp += i;
                  m \gg = 1 + i;
01730
                  sign ^= msign & 0x8000;
01731
                  if(exp < -11)
01732
                      return underflow<R>(sign);
                  m = divide64(m, L, s);
return fixed2half<R,30,false,false,true>(m, exp, sign, 1);
01733
01734
```

```
01735
              }
01736
01745
              template<std::float_round_style R> unsigned int hypot_post(uint32 r, int exp)
01746
                   int i = r \gg 31:
01747
01748
                  if((exp+=i) > 46)
01749
                       return overflow<R>();
01750
                  if(exp < -34)
                  return underflow<R>();
r = (r*i) | (r&i);
01751
01752
                  uint32 m = sqrt<30>(r, exp+=15);
01753
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;
                  if(exp > 29)
01771
01772
                       return overflow<R>(sign);
01773
                  if(exp < -11)</pre>
01774
                      return underflow<R>(sign);
                  uint32 m = divide64(myx(i+1), mx, s);
return fixed2half<R,30,false,false,true>(m, exp, sign, s);
01775
01776
01777
              }
01778
01788
              template<std::float_round_style R,bool S> unsigned int area(unsigned int arg)
01789
                   int abs = arg & 0x7FFF, expx = (abs > 10) + (abs <= 0x3FF) - 15, <math>expy = -15, ilog, i;
01790
                  uint32 mx = static_cast<uint32>((abs&0x3FF)|((abs>0x3FF) <10)) < 20, my, r;</pre>
01791
01792
                   for(; abs<0x400; abs<=1,--expy);</pre>
01793
                   expy += abs » 10;
01794
                  r = ((abs \& 0x3FF) | 0x400) \ll 5;
                  r *= r;
i = r » 31;
01795
01796
                  expy = 2*expy + i;
r >= i;
01797
01798
01799
                   if(S)
01800
01801
                       if(expy < 0)
01802
                       {
                           r = 0x40000000 + ((expy>-30))?
01803
      ((r»-expy) | ((r&((static_cast<uint32>(1) «-expy)-1))!=0)) : 1);
01804
                         expy = 0;
01805
01806
                       else
01807
                           r += 0x40000000 \Rightarrow expy;
01808
01809
                           i = r \gg 31;
                           r = (r*i) | (r&i);
01810
01811
                           expy += i;
01812
                       }
01813
01814
                  else
01815
                  {
                      r -= 0x40000000 » expy;
01816
                       for(; r<0x40000000; r«=1,--expy);</pre>
01817
01818
01819
                  my = sqrt < 30 > (r, expy);
                  my = (my < 15) + (r < 14) / my;
01820
01821
                  if(S)
01822
                  {
01823
                       mx »= expy - expx;
                      ilog = expy;
01824
01825
01826
                   else
01827
                  {
01828
                       my »= expx - expy;
01829
                      ilog = expx;
01830
01831
                   my += mx;
01832
                  i = my \gg 31;
                  static const int G = S && (R==std::round_to_nearest);
01833
                   return log2_post<R, 0xB8AA3B2A>(log2 (my»i, 26+S+G)+(G«3), ilog+i, 17,
01834
     arg&(static_cast<unsigned>(S) «15));
01835
              }
01836
01838
              struct f31
01839
                  HALF_CONSTEXPR f31(uint32 mant, int e) : m(mant), exp(e) {}
01843
01844
01847
                   f31(unsigned int abs) : exp(-15)
01848
01849
                       for(; abs<0x400; abs«=1,--exp);</pre>
                       01850
01851
                       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);
                       int d = a.exp - b.exp;
01862
                       uint32 m = a.m + ((d<32) ? (b.m>d) : 0);
01863
01864
                       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;
                       uint32 m = a.m - ((d<32) ? (b.m»d) : 0);
01875
01876
                       if(!m)
01877
                           return f31(0, -32);
                       for(; m<0x80000000; m<=1,--exp);</pre>
01878
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 \gg 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)), 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
                   unsigned int abs = arg & 0x7FFF, sign = arg & 0x8000;
01920
                   f31 \times (abs), \times2 = \times \times \times \times f31(0\times88AA3B29, 0), t = f31(0\times80000000, 0) / (f31(0\times80000000,
01921
      0) +f31(0xA7BA054A, -2)*x), t2 = t * t;
01922
                  f31 = ((f31(0x87DC2213, 0)*t2+f31(0xB5F0E2AE, 0))*t2+f31(0x82790637,
     01923
01924
      sign&(C-1U)) :
01925
                           (e.exp<-25) ? underflow<R>() : fixed2half<R,30,false,false,true>(e.m»1, e.exp+14,
      0, e.m&1);
01926
01927
01937
              template<std::float round style R, bool L> unsigned int gamma(unsigned int arg)
01938
                   static const double p[] ={ 2.50662827563479526904, 225.525584619175212544,
      -268.295973841304927459,\ 80.9030806934622512966,\ -5.00757863970517583837,\ 0.0114684895434781459556\ \};
01940
                  double t = arg + 4.65, s = p[0];
01941
                  for (unsigned int i=0; i<5; ++i)
                  s += p[i+1] / (arg+i);
return std::log(s) + (arg-0.5)*std::log(t)
01942
01943
01944 */
                   static const f31 pi(0xC90FDAA2, 1), lbe(0xB8AA3B29, 0);
01945
                   unsigned int abs = arg & 0x7FFF, sign = arg & 0x8000;
01946
                  bool bsign = sign != 0;
                  f31 z(abs), x = sign? (z+f31(0x80000000, 0)) : z, t = x + f31(0x94CCCCD, 2), s = f31(0x806C9901, 1) + f31(0xBBE654E2, <math>-7)/(x+f31(0x80000000, 2)) + f31(0xA1CE6098,
01947
01948
      6) /(x+f31(0x80000000, 1))
01949
                       + f31(0xE1868CB7, 7)/x - f31(0x8625E279, 8)/(x+f31(0x80000000, 0)) - f31(0xA03E158F,
     2)/(x+f31(0xC0000000, 1));
01950
                  int i = (s.exp>=2) + (s.exp>=4) + (s.exp>=8) + (s.exp>=16);
01951
                   s = f31((static_cast<uint32>(s.exp) ((31-i)) + (log2(s.m)1, 28)), i) / lbe;
01952
                   if(x.exp != -1 || x.m != 0x80000000)
01953
01954
                       i = (t.exp>=2) + (t.exp>=4) + (t.exp>=8);
01955
                       f31 l = f31((static_cast<uint32>(t.exp) ((31-i))+(log2(t.m»1, 30) »i), i) / lbe;
01956
                       s = (x.exp<-1) ? (s-(f31(0x80000000, -1)-x)*1) : (s+(x-f31(0x80000000, -1))*1);
01957
01958
                  s = x.exp ? (s-t) : (t-s):
01959
                   if (bsign)
01960
01961
                       if(z.exp >= 0)
01962
01963
                           sign &= (L|((z.m\gg(31-z.exp))&1)) - 1;
                           for(z=f31((z.m«(1+z.exp))&0xFFFFFFFF, -1); z.m<0x80000000; z.m«=1,--z.exp);</pre>
01964
01965
```

```
if(z.exp == -1)
01967
                          z = f31(0x80000000, 0) - z;
01968
                      if(z.exp < -1)
01969
01970
                          z = z * pi;
                          z.m = \frac{1}{\text{sincos}}(z.m)(1-z.exp), 30).first;
01971
01972
                          for(z.exp=1; z.m<0x80000000; z.m«=1,--z.exp);</pre>
01973
01974
                          z = f31(0x80000000, 0);
01975
01976
01977
                  if(L)
01978
                  {
01979
                      if(bsign)
01980
01981
                          f31 1(0x92868247, 0);
01982
                          if(z.exp < 0)
01983
                          {
01984
                              uint32 m = log2((z.m+1)), 27);
01985
                              z = f31(-((static_cast<uint32>(z.exp) < 26) + (m>5)), 5);
01986
                              for(; z.m<0x80000000; z.m«=1,--z.exp);</pre>
01987
                              1 = 1 + z / lbe;
01988
                          01989
01990
                          s = sign ? (s-1) : x.exp ? (1-s) : (1+s);
01991
                      }
01992
                      else
01993
                          sign = static_cast<unsigned>(x.exp==0) « 15;
01994
                          if(s.exp < -24)
01995
                             return underflow<R>(sign);
01996
01997
                          if(s.exp > 15)
01998
                             return overflow<R>(sign);
01999
                      }
02000
02001
                  else
02002
02003
                      s = s * lbe;
                      uint32 m;
02004
02005
                      if(s.exp < 0)
02006
02007
                          m = s.m \gg -s.exp;
                          s.exp = 0;
02008
02009
02010
                      else
02011
02012
                          m = (s.m \ll 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;

s = pi / s;
02022
02023
                          if(s.exp < -24)
02024
                              return underflow<R>(sign);
02025
02026
                      else if (z.exp > 0 && !(z.m&((1 < (31-z.exp))-1)))
                      return ((s.exp+14) <10) + (s.m>21);
if(s.exp > 15)
02027
02028
02029
                          return overflow<R>(sign);
02030
02031
                  return fixed2half<R,31,false,false,true>(s.m, s.exp+14, sign);
02032
02034
02035
              template<typename,typename,std::float_round_style> struct half_caster;
02036
         }
02037
02055
          class half
02056
         public:
02057
02060
02064
              HALF_CONSTEXPR half() HALF_NOEXCEPT : data_() {}
02065
02069
              explicit half(float rhs) :
     data_(static_cast<detail::uint16>(detail::float2half<round_style>(rhs))) {}
02070
02073
              operator float() const { return detail::half2float<float>(data ); }
02074
              half& operator=(float rhs) { data_ =
     static_cast<detail::uint16>(detail::float2half<round_style>(rhs)); return *this; }
02080
02084
02090
             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
               half& operator/=(half rhs) { return *this = *this / rhs; }
02111
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
               half& operator++() { return *this = *this + half(detail::binary, 0x3C00); }
02144
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
               HALF_CONSTEXPR half(detail::binary_t, unsigned int bits) HALF_NOEXCEPT :
02168
     data_(static_cast<detail::uint16>(bits)) {}
02169
02171
               detail::uint16 data :
02172
02173
           #ifndef HALF_DOXYGEN_ONLY
02174
               friend HALF_CONSTEXPR_NOERR bool operator==(half, half);
               friend HALF_CONSTEXPR_NOERR bool operator!=(half, half);
02175
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);
               friend half operator+(half, half);
friend half operator-(half, half);
02181
02182
               friend half operator*(half, half);
02183
02184
               friend half operator/(half, half);
02185
               template<typename charT,typename traits> friend std::basic_ostream<charT,traits>&
      operator«(std::basic_ostream<charT,traits>&, half);
02186
               template<typename charT,typename traits> friend std::basic_istream<charT,traits>&
      operator»(std::basic_istream<charT,traits>&, half&);
02187
               friend HALF_CONSTEXPR half fabs(half);
friend half fmod(half, half);
02188
02189
               friend half remainder (half, half);
02190
               friend half remquo(half, half, int*);
02191
               friend half fma(half, half, half);
               friend HALF_CONSTEXPR_NOERR half fmax(half, half);
friend HALF_CONSTEXPR_NOERR half fmin(half, half);
02192
02193
               friend half fdim(half, half);
friend half nanh(const char*);
02194
02195
               friend half exp(half);
02196
02197
               friend half exp2(half);
02198
               friend half expm1(half);
               friend half log(half);
friend half log10(half);
02199
02200
02201
               friend half log2(half);
               friend half log1p(half);
02202
02203
               friend half sqrt(half);
02204
               friend half rsqrt(half);
               friend half cbrt(half);
02205
02206
               friend half hypot (half, half);
02207
               friend half hypot (half, half, half);
02208
               friend half pow(half, half);
02209
               friend void sincos(half, half*, half*);
02210
               friend half sin(half);
02211
               friend half cos(half);
02212
               friend half tan(half):
02213
               friend half asin(half);
02214
               friend half acos(half);
02215
               friend half atan(half);
02216
               friend half atan2(half, half);
02217
               friend half sinh(half):
02218
               friend half cosh(half);
               friend half tanh(half);
02219
02220
               friend half asinh(half);
               friend half acosh (half);
02221
02222
               friend half atanh (half);
02223
               friend half erf(half);
               friend half erfc(half);
friend half lgamma(half);
02224
02225
```

```
friend half tgamma(half);
              friend half ceil(half);
02227
02228
              friend half floor(half);
02229
              friend half trunc(half);
02230
              friend half round(half):
02231
              friend long lround(half);
              friend half rint(half);
              friend long lrint(half);
02233
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 scalbln(half, long);
02241
              friend half modf(half, half*);
             friend int ilogb(half);
02242
              friend half logb(half);
friend half nextafter(half, half);
02243
              friend half nexttoward(half, long double);
02245
02246
              friend HALF_CONSTEXPR half copysign(half, half);
02247
              friend HALF_CONSTEXPR int fpclassify(half);
02248
              friend HALF_CONSTEXPR bool isfinite(half);
              friend HALF_CONSTEXPR bool isinf(half);
02249
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
              inline half operator "" _h(long double value) { return half(detail::binary,
02279
     detail::float2half<half::round_style>(value)); }
02280
02281 #endif
02283
          namespace detail
02284
         {
     template<typename T,typename U,std::float_round_style
R=(std::float_round_style)(HALF_ROUND_STYLE)> struct half_caster {};
02291
02292
              template<typename U, std::float round style R> struct half caster<half, U, R>
02293
02294
              #if HALF_ENABLE_CPP11_STATIC_ASSERT && HALF_ENABLE_CPP11_TYPE_TRAITS
                  static_assert(std::is_arithmetic<U>::value, "half_cast from non-arithmetic type
02295
     unsupported");
02296
              #endif
02297
02298
                  static half cast(U arg) { return cast_impl(arg, is_float<U>()); };
02299
02300
              private:
02301
                  static half cast_impl(U arg, true_type) { return half(binary, float2half<R>(arg)); }
                  static half cast_impl(U arg, false_type) { return half(binary, int2half<R>(arg)); }
02302
02303
              };
02304
              template<typename T.std::float round style R> struct half caster<T.half.R>
02305
02306
              #if HALF_ENABLE_CPP11_STATIC_ASSERT && HALF_ENABLE_CPP11_TYPE_TRAITS
02307
                  static_assert(std::is_arithmetic<T>::value, "half_cast to non-arithmetic type
     unsupported");
02308
              #endif
02309
02310
                  static T cast(half arg) { return cast_impl(arg, is_float<T>()); }
02311
02312
02313
                  static T cast_impl(half arg, true_type) { return half2float<T>(arg.data_); }
02314
                  static T cast impl(half arg, false type) { return half2int<R,true,true,T>(arg.data ); }
02315
02316
              template<std::float_round_style R> struct half_caster<half,half,R>
02317
02318
                  static half cast(half arg) { return arg; }
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
              static HALF CONSTEXPR CONST bool is integer = false;
02338
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
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
              static HALF CONSTEXPR CONST int max exponent10 = 4;
02402
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
              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
          template<> struct hash<half float::half>
02438
```

```
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
02466
                 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)) <
02491
          ((y.data_^(0x8000|(0x8000-(y.data_»15)))))+(y.data_»15));
02492
                }
02493
02500
                 inline HALF_CONSTEXPR_NOERR bool operator>(half x, half y)
02501
                        return !detail::compsignal(x.data_, y.data_) &&
    ((x.data_^(0x8000|(0x8000-(x.data_*15))))+(x.data_*15))) >
02502
02503
          ((y.data_^(0x8000|(0x8000-(y.data_»15))))+(y.data_»15));
02504
                 }
02505
02512
                 inline HALF_CONSTEXPR_NOERR bool operator<=(half x, half y)</pre>
02513
                        return !detail::compsignal(x.data_, y.data_) &&
02514
02515
                              ((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, v.data) &&
                              ((x.data_^(0x8000|(0x8000-(x.data_*15))))+(x.data_*15))) >=
02527
          ((y.data_^(0x8000|(0x8000-(y.data_»15))))+(y.data_»15));
02528
02529
02534
02538
                 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::internal_t>(x.data_)+detail::half2float<detail::internal_t>(x.data_)+detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<de
02556
02557
                       int absx = x.data_ & 0x7FFF, absy = y.data_ & 0x7FFF;
                       if (absx >= 0x7c00 || absy >= 0x7c00)

if (absx >= 0x7c00 || absy >= 0x7c00)
02558
02559
                              return half (detail::binary,
                                                                               (absx>0x7C00 || absy>0x7C00) ? detail::signal(x.data_,
02560
         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(!absv)
02565
                              return x;
02566
                        unsigned int sign = ((sub && absy>absx) ? y.data_ : x.data_) & 0x8000;
02567
                       if(absy > absx)
02568
                              std::swap(absx, absy);
                        int exp = (absx*10) + (absx<=0x3FF), d = exp - (absy*10) - (absy<=0x3FF), mx = absx
02569
         ((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
```

```
my = 1;
 02577
                                         if(sub)
02578
02579
                                                     if(!(mx-=my))
02580
                                                                return half (detail::binary,
                 static_cast<unsiqned>(half::round_style==std::round_toward_neq_infinity) <15);</pre>
 02581
                                                  for(; mx<0x2000 && exp>1; mx«=1,--exp);
02582
02583
                                         else
02584
                                         {
02585
                                                    mx += my;
02586
                                                    int i = mx \gg 14;
 02587
                                                    if((exp+=i) > 30)
 02588
                                                                return half(detail::binary, detail::overflow<half::round_style>(sign));
02589
                                                    mx = (mx \gg i) \mid (mx \& i);
02590
02591
                                         return half (detail::binary, detail::rounded<half::round style,false>(sign+((exp-1) < 10) + (mx > 3),
                 (mx \times 2) \& 1, (mx \& 0x3) !=0);
                            #endif
 02592
 02593
 02594
02602
                             inline half operator-(half x, half y)
02603
                              #ifdef HALF ARITHMETIC TYPE
02604
                                         return half (detail::binary,
02605
                 detail::float2half<half::round_style>(detail::half2float<detail::internal_t>(x.data_)-detail::half2float<detail::internal_t>(x.data_)-detail::half2float<detail::internal_t>(x.data_)-detail::half2float<detail::internal_t>(x.data_)-detail::half2float<detail::internal_t>(x.data_)-detail::half2float<detail::internal_t>(x.data_)-detail::half2float<detail::internal_t>(x.data_)-detail::half2float<detail::internal_t>(x.data_)-detail::half2float<detail::internal_t>(x.data_)-detail::half2float<detail::internal_t>(x.data_)-detail::half2float<detail::internal_t>(x.data_)-detail::half2float<detail::internal_t>(x.data_)-detail::half2float<detail::internal_t>(x.data_)-detail::half2float<detail::internal_t>(x.data_)-detail::half2float<detail::internal_t>(x.data_)-detail::half2float<detail::internal_t>(x.data_)-detail::half2float<detail::internal_t>(x.data_)-detail::half2float<detail::internal_t>(x.data_)-detail::half2float<detail::internal_t>(x.data_)-detail::half2float<detail::internal_t>(x.data_)-detail::half2float<detail::internal_t>(x.data_)-detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half
02606
02607
                                        return x + -y;
02608
                              #endif
02609
02610
 02618
                             inline half operator* (half x, half y)
 02619
02620
                              #ifdef HALF_ARITHMETIC_TYPE
                                         return half(detail::binary,
02621
                 detail::float2half<half::round_style>(detail::half2float<detail::internal_t>(x.data_)*detail::half2float<detail::internal_t>(x.data_)*detail::half2float<detail::internal_t>(x.data_)*detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<de
02622
                             #else
02623
                                        int absx = x.data_ & 0x7FFF, absy = y.data_ & 0x7FFF, exp = -16;
                                         unsigned int sign = (x.data_^y.data_) & 0x8000;
if(absx >= 0x7C00 || absy >= 0x7C00)
 02624
 02625
02626
                                                     return half(detail::binary,
                                                                                                                                            (absx>0x7C00 || absy>0x7C00) ? detail::signal(x.data_,
                 y.data_) :
02627
                                                                                                                                        ((absx==0x7C00 \&\& !absv)) | (absv==0x7C00 \&\& !absx))?
                detail::invalid() : (sign|0x7C00));
 02628
                                       if(!absx || !absy)
02629
                                                     return half (detail::binary, sign);
02630
                                         for(; absx<0x400; absx«=1,--exp);</pre>
                                         for(; absy<0x400; absy«=1,--exp);</pre>
02631
                                        detail::uint32 m = static cast<detail::uint32>((absx&0x3FF)|0x400) *
02632
                static_cast<detail::uint32>((absy&0x3FF)|0x400);
 02633
                                         int i = m \gg 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)
                                         return half(detail::binary, detail::underflow<half::round_style>(sign));
return half(detail::binary, detail::fixed2half<half::round_style,20,false,false,false>(m»i,
 02638
 02639
                 exp, sign, s));
 02640
                              #endif
02641
                             }
02642
02651
                             inline half operator/(half x, half v)
 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::internal_t>(x.data_)/detail::half2float<detail::internal_t>(x.data_)/detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail:half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<detail::half2float<det
02655
                             #else
                                         int absx = x.data_ & 0x7FFF, absy = y.data_ & 0x7FFF, exp = 14;
02656
                                         unsigned int sign = (x.data_^y.data_) & 0x8000;
02657
                                         if(absx >= 0x7C00 \mid \mid absy >= 0x7C00)
 02658
                                                    return half (detail::binary,
                                                                                                                                           (absx>0x7C00 || absy>0x7C00) ? detail::signal(x.data_,
02659
                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) ;</pre>
                                         for(; absy<0x400; absy«=1,++exp);</pre>
02666
                                         detail::uint32 mx = (absx&0x3FF) | 0x400, my = (absy&0x3FF) | 0x400;
02667
 02668
                                         int i = mx < my;
                                         exp += (absx*10) - (absy*10) - i;
 02669
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 \ll 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
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_&0x7FFF); }
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_ & 0x7FFF, absy = y.data_ & 0x7FFF, sign = x.data_ & 0x8000;
02746
              if(absx \ge 0x7C00 \mid \mid absy \ge 0x7C00)
                  return half(detail::binary,
02747
                                                  (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)
                  return half(detail::binary, sign);
02754
02755
              return half(detail::binary, sign|detail::mod<false,false>(absx, absy));
02756
          }
02757
02764
          inline half remainder (half x, half y)
02765
              unsigned int absx = x.data_ & 0x7FFF, absy = y.data_ & 0x7FFF, sign = x.data_ & 0x8000;
02766
02767
              if(absx >= 0x7C00 \mid \mid absy >= 0x7C00)
                  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 == absv)
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_ & 0x7FFF, absy = y.data_ & 0x7FFF, value = x.data_ & 0x8000;
              if(absx >= 0x7C00 \mid \mid absy >= 0x7C00)
02787
                 return half(detail::binary, (absx>0x7C00 || absy>0x7C00) ? detail::signal(x.data_,
02788
     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;
              if(absx != absy)
  value ^= detail::mod<true, true>(absx, absy, &q);
02794
02795
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
      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_);
02812
```

```
#if HALF_ENABLE_CPP11_CMATH && FP_FAST_FMA
                   return half(detail::binary, detail::float2half<half::round_style>(std::fma(fx, fy, fz)));
02814
02815
               #else
02816
                   return half(detail::binary, detail::float2half<half::round_style>(fx*fy+fz));
02817
               #endif
02818
          #else
              int absx = x.data_ & 0x7FFF, absy = y.data_ & 0x7FFF, absz = z.data_ & 0x7FFF, exp = -15; unsigned int sign = (x.data_^y.data_) & 0x8000;
02820
02821
               bool sub = ((sign^z.data_)&0x8000) != 0;
               if(absx >= 0x7C00 || absy >= 0x7C00 || absz >= 0x7C00)
    return (absx>0x7C00 || absy>0x7C00 || absz>0x7C00) ? half(detail::binary,
02822
02823
      02824
      detail::invalid() : (sign|0x7C00)) :
02825
                            (absy==0x7C00) ? half(detail::binary, (!absx || (sub && absz==0x7C00)) ?
      detail::invalid() : (sign|0x7C00)) : z;
02826
              if(!absx || !absy)
      return abso? ? z : half(detail::binary, (half::round_style==std::round_toward_neg_infinity) ? (z.data_|sign) : (z.data_&sign));
02827
02828
              for(; absx<0x400; absx«=1,--exp);</pre>
               for(; absy<0x400; absy<=1,--exp);
detail::uint32 m = static_cast<detail::uint32>((absx&0x3FF)|0x400) *
02829
02830
      static_cast<detail::uint32>((absy&0x3FF)|0x400);
02831
              int. i = m \gg 21:
02832
               exp += (absx*10) + (absy*10) + i;
               m «= 3 - i;
02834
               if(absz)
02835
               {
02836
                   int expz = 0;
                   for(; absz<0x400; absz«=1,--expz) ;
expz += absz » 10;</pre>
02837
02838
02839
                   detail::uint32 mz = static_cast<detail::uint32>((absz&0x3FF)|0x400) « 13;
02840
                   if(expz > exp \mid \mid (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;
                   if(sub)
02849
02850
                   {
02851
                       m = m - mz;
02852
                       <u>if</u>(!m)
02853
                            return half(detail::binary,
      static_cast<unsigned>(half::round_style==std::round_toward_neg_infinity) <15);</pre>
02854
                       for(; m<0x800000; m<=1,--exp);</pre>
02855
                   }
02856
                   else
                   {
02858
                       m += mz;
02859
                       i = m \gg 24;
02860
                       m = (m \approx i) \mid (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)
                  return half(detail::binary, detail::underflow<half::round_style>(sign));
02867
               return half(detail::binary, detail::fixed2half<half::round_style,23,false,false,false>(m,
02868
      exp-1, sign));
02869
           #endif
02870
02871
02878
          inline HALF_CONSTEXPR_NOERR half fmax(half x, half y)
02879
               return half(detail::binary, (!isnan(y) && (isnan(x) ||
02880
      (x.data_^(0x8000|(0x8000-(x.data_»15)))) <
02881
                   (y.data_^(0x8000|(0x8000-(y.data_*x15))))))) ? detail::select(y.data_, x.data_) :
      detail::select(x.data_, y.data_));
02882
          }
02883
02890
           inline HALF_CONSTEXPR_NOERR half fmin(half x, half y)
02891
               return half(detail::binary, (!isnan(y) && (isnan(x) ||
02892
      (x.data_^(0x8000|(0x8000-(x.data_*15))))) >
02893
                  (y.data_^(0x8000|(0x8000-(y.data_>15)))))) ? detail::select(y.data_, x.data_) :
      detail::select(x.data_, y.data_));
02894
          }
02895
02904
           inline half fdim(half x, half y)
02905
02906
               if(isnan(x) || isnan(y))
                   return half(detail::binary, detail::signal(x.data_, y.data_));
02907
               return (x.data_^(0x8000|(0x8000-(x.data_w15))))) <= (y.data_^(0x8000|(0x8000-(y.data_w15)))) ?
02908
```

```
half(detail::binary, 0) : (x-y);
02909
02910
02915
          inline half nanh(const char *arg)
02916
              unsigned int value = 0x7FFF;
02917
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
              return half(detail::binary,
02939
     detail::float2half<half::round_style>(std::exp(detail::half2float<detail::internal_t>(arg.data_))));
02940
          #else
02941
             int abs = arg.data_ & 0x7FFF, e = (abs\approx10) + (abs<=0x3FF), exp;
02942
              if(!abs)
02943
                  return half(detail::binary, 0x3C00);
02944
              if(abs >= 0x7C00)
                 return half(detail::binary, (abs==0x7C00) ? (0x7C00&((arg.data_>15)-1U)) :
02945
     detail::signal(arg.data_));
02946
             if(abs >= 0x4C80)
                  return half(detail::binary, (arg.data_&0x8000) ? detail::underflow<half::round_style>() :
02947
     detail::overflow<half::round_style>());
02948
              detail::uint32 m =
     detail::multiply64(static_cast<detail::uint32>((abs&0x3FF)+((abs>0x3FF) &10)) &21, 0xB8AA3B29);
02949
              if(e < 14)
02950
              {
02951
                  exp = 0;
02952
                  m »= 14 - e;
02953
02954
              else
02955
              {
02956
                  exp = m \gg (45-e);
02957
                  m = (m \ll (e-14)) \& 0 \times 7 FFFFFFF;
02958
02959
              return half(detail::binary, detail::exp2_post<half::round_style>(m, exp,
      (arg.data_&0x8000)!=0, 0, 26));
02960
          #endif
02961
          }
02962
02971
          inline half exp2(half arg)
02972
02973
          #if defined(HALF_ARITHMETIC_TYPE) && HALF_ENABLE_CPP11_CMATH
              return half (detail::binary,
02974
     detail::float2half<half::round_style>(std::exp2(detail::half2float<detail::internal_t>(arg.data_))));
02975
         #else
02976
              int abs = arg.data_ & 0x7FFF, e = (abs\approx10) + (abs<=0x3FF), exp = (abs\approx0x3FF) +
      ((abs>0x3FF)«10);
02977
              if(!abs)
02978
                  return half(detail::binary, 0x3C00);
02979
              if(abs >= 0x7C00)
                  return half(detail::binary, (abs==0x7C00) ? (0x7C00&((arg.data_*)15)-1U)) :
02980
     detail::signal(arg.data_));
02981
              if(abs >= 0x4E40)
                  return half(detail::binary, (arg.data_&0x8000) ? detail::underflow<half::round_style>() :
02982
     detail::overflow<half::round_style>());
            return half(detail::binary, detail::exp2_post<half::round_style>(
02983
                  (static_cast<detail::uint32>(exp) «(6+e)) &0x7FFFFFFF, exp» (25-e), (arg.data_&0x8000)!=0, 0,
02984
     28));
02985
          #endif
02986
02987
02997
          inline half expml(half arg)
02998
02999
          #if defined(HALF_ARITHMETIC_TYPE) && HALF_ENABLE_CPP11_CMATH
03000
              return half (detail::binary,
      detail::float2half<half::round_style>(std::expm1(detail::half2float<detail::internal_t>(arg.data_))));
03001
03002
              unsigned int abs = arg.data_ & 0x7FFF, sign = arg.data_ & 0x8000, e = (abs>10) + (abs<=0x3FF),
      exp;
03003
              if(!abs)
03004
                  return arg;
03005
              if(abs >= 0x7C00)
03006
                  return half(detail::binary, (abs==0x7C00) ? (0x7C00+(sign»1)) :
      detail::signal(arg.data_));
03007
              if(abs >= 0x4A00)
                  return half(detail::binary, (arg.data_&0x8000) ?
03008
     detail::rounded<half::round_style,true>(0xBBFF, 1, 1) : detail::overflow<half::round_style>());
              detail::uint32 m =
03009
     detail::multiply64(static_cast<detail::uint32>((abs&0x3FF)+((abs>0x3FF) &10)) &21, 0xB8AA3B29);
03010
              if(e < 14)
03011
              {
03012
                  exp = 0;
```

```
03013
                   m \gg = 14 - e;
03014
03015
               else
03016
               {
03017
                  exp = m \gg (45-e);

m = (m \ll (e-14)) \& 0x7FFFFFFF;
03018
03019
03020
               m = detail::exp2(m);
03021
               if(sign)
03022
               {
                   int s = 0:
03023
                   if(m > 0x80000000)
03024
03025
                   {
03026
03027
                       m = detail::divide64(0x80000000, m, s);
03028
                   m = 0x80000000 - ((m»exp) | ((m&((static_cast<detail::uint32>(1) «exp)-1))!=0)|s);
03029
03030
                   exp = 0;
03031
               }
03032
               else
03033
                   m = (exp<31) ? (0x80000000wexp) : 1;
03034
               for(exp+=14; m<0x80000000 && exp; m«=1,--exp);</pre>
03035
               if(exp > 29)
               return half(detail::binary, detail::overflow<half::round_style>());
return half(detail::binary, detail::rounded<half::round_style,true>(sign+(exp«10)+(m»21),
03036
03037
      (m>20)&1, (m&0xFFFFF)!=0));
03038
           #endif
03039
03040
03050
           inline half log(half arg)
03051
03052
           #ifdef HALF_ARITHMETIC_TYPE
               return half (detail::binary,
03053
      detail::float2half<half::round_style>(std::log(detail::half2float<detail::internal_t>(arg.data_))));
03054
           #else
03055
               int abs = arg.data_ & 0x7FFF, exp = -15;
03056
               if(!abs)
03057
                   return half(detail::binary, detail::pole(0x8000));
03058
               if(arg.data_ & 0x8000)
                   return half(detail::binary, (arg.data_<=0xFC00) ? detail::invalid() :</pre>
03059
      detail::signal(arg.data_));
03060
              if(abs >= 0x7C00)
                   return (abs==0x7C00) ? arg : half(detail::binary, detail::signal(arg.data_));
03061
03062
               for(; abs<0x400; abs«=1,--exp);</pre>
               exp += abs » 10;
03063
03064
               return half(detail::binary, detail::log2_post<half::round_style,0xB8AA3B2A>(
03065
                  detail::log2(static_cast<detail::uint32>((abs&0x3FF)|0x400) < 20, 27) + 8, exp, 17));</pre>
03066
           #endif
03067
          }
03068
03078
           inline half log10 (half arg)
03079
03080
           #ifdef HALF_ARITHMETIC_TYPE
      return half(detail::binary,
detail::float2half<half::round_style>(std::log10(detail::half2float<detail::internal_t>(arg.data_))));
03081
03082
           #else
03083
              int abs = arg.data_ & 0x7FFF, exp = -15;
03084
               if(!abs)
03085
                   return half(detail::binary, detail::pole(0x8000));
03086
               if(arg.data_ & 0x8000)
03087
                   return half(detail::binary, (arg.data <=0xFC00) ? detail::invalid() :</pre>
      detail::signal(arg.data_));
    if(abs >= 0x7C00)
03088
03089
                   return (abs==0x7C00) ? arg : half(detail::binary, detail::signal(arg.data_));
03090
               switch(abs)
03091
               {
03092
                   case 0x4900: return half(detail::binary, 0x3C00);
03093
                   case 0x5640: return half(detail::binary, 0x4000);
03094
                   case 0x63D0: return half(detail::binary, 0x4200);
03095
                   case 0x70E2: return half(detail::binary, 0x4400);
03096
03097
               for(; abs<0x400; abs«=1,--exp);</pre>
03098
               exp += abs \gg 10;
               return half(detail::binary, detail::log2_post<half::round_style,0xD49A784C>(
03099
03100
                   detail::log2(static_cast<detail::uint32>((abs&0x3FF)|0x400) < 20, 27) + 8, exp, 16));</pre>
03101
           #endif
03102
03103
03113
           inline half log2(half arg)
0.3114
           #if defined(HALF ARITHMETIC TYPE) && HALF ENABLE CPP11 CMATH
03115
03116
               return half(detail::binary,
      detail::float2half<half::round_style>(std::log2(detail::half2float<detail::internal_t>(arg.data_))));
03117
           #else
03118
              int abs = arg.data_ & 0x7FFF, exp = -15, s = 0;
0.3119
               if(!abs)
                   return half (detail::binary, detail::pole(0x8000));
03120
```

```
if(arg.data_ & 0x8000)
                  return half(detail::binary, (arg.data_<=0xFC00) ? detail::invalid() :</pre>
      detail::signal(arg.data_));
03123
              if(abs >= 0x7C00)
                  return (abs==0x7C00) ? arg : half(detail::binary, detail::signal(arg.data_));
03124
              if(abs == 0x3C00)
03125
03126
                  return half(detail::binary, 0);
03127
              for(; abs<0x400; abs<=1,--exp);</pre>
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);</pre>
                   return half(detail::binary, value+(exp«10)+m);
03133
03134
0.3135
              detail::uint32 ilog = exp, sign = detail::sign_mask(ilog), m =
                  03136
      - sign;
03137
              <u>if</u>(!m)
03138
                   return half(detail::binary, 0);
03139
               for (exp=14; m<0x8000000 && exp; m<=1,--exp);</pre>
03140
              for(; m>0xFFFFFFF; m>=1,++exp)
0.3141
                  s |= m & 1;
               return half (detail::binary, detail::fixed2half<half::round style,27,false,false,true>(m, exp,
0.3142
     sign&0x8000, s));
03143
         #endif
03144
03145
03156
          inline half log1p(half arg)
03157
           #if defined(HALF_ARITHMETIC_TYPE) && HALF_ENABLE_CPP11_CMATH
03158
03159
              return half (detail::binary,
      detail::float2half<half::round_style>(std::log1p(detail::half2float<detail::internal_t>(arg.data_))));
03160
03161
              if(arg.data_ >= 0xBC00)
      return half(detail::binary, (arg.data_==0xBC00) ? detail::pole(0x8000) : (arg.data_<=0xFC00) ? detail::invalid() : detail::signal(arg.data_));
03162
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) ;</pre>
0.3167
              exp += abs \gg 10;
              detail::uint32 m = static cast<detail::uint32>((abs&0x3FF)|0x400) « 20;
03168
               if(arg.data_ & 0x8000)
03169
03170
              {
                   m = 0x40000000 - (m - exp);
03171
03172
                   for(exp=0; m<0x40000000; m<=1,--exp);</pre>
03173
03174
              else
03175
03176
                   if(exp < 0)
03177
0.3178
                       m = 0x40000000 + (m - exp);
03179
                       exp = 0;
03180
03181
                  else
03182
03183
                       m += 0x40000000 \gg exp;
03184
                       int i = m \gg 31;
03185
                       m \gg = 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
0.3191
          }
03192
03197
03206
          inline half sqrt(half arg)
03207
03208
           #ifdef HALF ARITHMETIC TYPE
     return half(detail::binary,
detail::float2half<half::round_style>(std::sqrt(detail::half2float<detail::internal_t>(arg.data_))));
03209
03210
          #else
03211
              int abs = arg.data_ & 0x7FFF, exp = 15;
              if(!abs || arg.data_ >= 0x7C00)
    return half(detail::binary, (abs>0x7C00) ? detail::signal(arg.data_) : (arg.data_>0x8000)
03212
03213
      ? detail::invalid() : arg.data_);
              for(; abs<0x400; abs<=1,--exp) ;
detail::uint32 r = static_cast<detail::uint32>((abs&0x3FF)|0x400) « 10, m =
03214
03215
      detail::sqrt<20>(r, exp+=abs>10);
              return half(detail::binary, detail::rounded<half::round_style,false>((exp«10)+(m&0x3FF), r>m,
      r!=0));
03217
          #endif
03218
          }
03219
```

```
03227
                 inline half rsqrt(half arg)
03228
03229
                 #ifdef HALF_ARITHMETIC_TYPE
03230
                       return half(detail::binary,
         detail::float2half<half::round style>(detail::internal t(1)/std::sgrt(detail::half2float<detail::internal t>(arg.data)
03231
                 #else
03232
                       unsigned int abs = arg.data_ & 0x7FFF, bias = 0x4000;
03233
                        if(!abs || arg.data_ >= 0x7C00)
                              return half(detail::binary,
                                                                                 (abs>0x7C00) ? detail::signal(arg.data_) :
03234
          (arg.data_>0x8000) ?
03235
                                                                             detail::invalid() : !abs ? detail::pole(arg.data &0x8000) :
         0);
03236
                        for(; abs<0x400; abs<=1,bias==0x400);</pre>
                       unsigned int frac = (abs+=bias) & 0x7FF;
03237
03238
                        if(frac == 0x400)
03239
                               return half(detail::binary, 0x7A00-(abs>1));
                        if((half::round_style == std::round_to_nearest && (frac == 0x3FE || frac == 0x76C)) ||
03240
                            (half::round_style != std::round_to_nearest && (frac == 0x15A || frac == 0x3FC || frac ==
03241
          0x401 \mid | frac == 0x402 \mid | frac == 0x67B)))
03242
                              return pow(arg, half(detail::binary, 0xB800));
                       detail::uint32 f = 0x17376 - abs, mx = (abs&0x3FF) | 0x400, my = ((f*x1)&0x3FF) | 0x400, mz = (abs&0x3FF) | 0x400, mz = 
03243
         my * my;
03244
                       int expy = (f \gg 11) - 31, expx = 32 - (abs \gg 10), i = mz \gg 21;
                       for(mz=0x60000000-(((mz»i)*mx)»(expx-2*expy-i)); mz<0x40000000; mz«=1,--expy);
03245
03246
                       i = (my*=mz*10) * 31;
03247
                       expy += i;
                       my = (my * (20+i)) + 1;
03248
                       i = (mz=my*my) \gg 21;
03249
03250
                       03251
                       i = (my *= (mz *10) +1) * 31;
                       return half(detail::binary, detail::fixed2half<half::round_style,30,false,false,true>(mywi,
03252
         expy+i+14));
03253
                 #endif
03254
03255
                 inline half cbrt(half arg)
03264
03265
03266
                 #if defined(HALF_ARITHMETIC_TYPE) && HALF_ENABLE_CPP11_CMATH
03267
                       return half(detail::binary,
         detail::float2half<half::round_style>(std::cbrt(detail::half2float<detail::internal_t>(arg.data_))));
03268
03269
                       int abs = arg.data_ & 0x7FFF, exp = -15;
                       if(!abs || abs == 0x3C00 || abs >= 0x7C00)
03270
03271
                              return (abs>0x7C00) ? half(detail::binary, detail::signal(arg.data_)) : arg;
                        for(; abs<0x400; abs«=1, --exp);</pre>
03272
03273
                        detail::uint32 ilog = exp + (abs>10), sign = detail::sign_mask(ilog), f, m =
03274
                             (((ilog«27)+(detail::log2(static_cast<detail::uint32>((abs&0x3FF)|0x400)«20, 24)»4))^sign)
         - sign;
03275
                       for (exp=2; m<0x80000000; m<=1,--exp);
03276
                       m = detail::multiply64(m, 0xAAAAAAAB);
                       int i = m \gg 31, s;
03278
                        exp += i;
03279
                       m \ll 1 - i;
03280
                        if(exp < 0)
03281
                       {
03282
                              f = m \gg -exp;
                              exp = 0;
03283
03284
03285
                       else
03286
                              f = (m \ll exp) \& 0x7FFFFFFF;
03287
03288
                             exp = m \gg (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
                              exp = -exp;
03298
03299
                       return half(detail::binary, (half::round_style==std::round_to_nearest) ?
    detail::fixed2half<half::round_style,31,false,false,false>(m, exp+14, arg.data_&0x8000) :
    detail::fixed2half<half::round_style,23,false,false,false>((m+0x80)»8, exp+14,
03300
03301
         arg.data_&0x8000));
03303
                 #endif
03304
03305
                 inline half hypot(half x, half y)
03315
03316
                 #ifdef HALF_ARITHMETIC_TYPE
03317
03318
                       detail::internal_t fx = detail::half2float<detail::internal_t>(x.data_), fy =
         detail::half2float<detail::internal_t>(y.data_);
03319
                       #if HALF_ENABLE_CPP11_CMATH
03320
                              return half(detail::binary, detail::float2half<half::round_style>(std::hypot(fx, fy)));
```

```
#else
                                           return half (detail::binary,
              detail::float2half<half::round_style>(std::sqrt(fx*fx+fy*fy)));
03323
                                 #endif
03324
                         #else
                                  int absx = x.data_ & 0x7FFF, absy = y.data_ & 0x7FFF, expx = 0, expy = 0; if(absx >= 0x7C00 || absy >= 0x7C00)
03325
03326
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)
                                           return half(detail::binary, absy ? detail::check_underflow(absy) : 0);
03330
                                  if(!absy)
03331
03332
                                            return half(detail::binary, detail::check_underflow(absx));
03333
                                   if(absy > absx)
03334
                                           std::swap(absx, absy);
                                  for(; absx<0x400; absx«=1,--expx);
for(; absy<0x400; absy«=1,--expy);</pre>
03335
03336
                                  detail::uint32 mx = (absx\&0x3FF) | 0x400, my = (absy\&0x3FF) | 0x400;
03337
03338
                                  mx \star = mx;
03339
                                  my *= my;
03340
                                  int ix = mx \gg 21, iy = my \gg 21;
03341
                                  expx = 2*(expx+(absx*10)) - 15 + ix;
                                  expy = 2*(expy+(absy*10)) - 15 + iy;
03342
03343
                                  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
                                  {\tt detail::internal\_t \ fx = detail::half2float < detail::internal\_t > (x.data\_), \ fy = detai
              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
                                 int absx = x.data_ & 0x7FFF, absy = y.data_ & 0x7FFF, absz = z.data_ & 0x7FFF, expx = 0, expy
03367
              = 0, expz = 0;
                                if(!absx)
03368
03369
                                           return hypot(y, z);
03370
                                  if(!absy)
03371
                                           return hypot(x, z);
03372
                                  if(!absz)
03373
                                           return hypot(x, y);
                                  if(absx >= 0x7C00 || absy >= 0x7C00 || absz >= 0x7C00)
   return half(detail::binary, (absx==0x7C00) ? det
03374
                                                                                                                    (absx==0x7C00) ? detail::select(0x7C00,
03375
              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 > absv)
03380
                                           std::swap(absy, absz);
03381
                                   if(absy > absx)
03382
                                           std::swap(absx, absy);
03383
                                  if(absz > absy)
03384
                                           std::swap(absv, absz);
                                  for(; absx<0x400; absx«=1,--expx);
for(; absy<0x400; absy«=1,--expy);</pre>
03385
03386
                                   for(; absz<0x400; absz<=1,--expz);</pre>
03387
03388
                                  detail::uint32 mx = (absx&0x3FF) | 0x400, my = (absy&0x3FF) | 0x400, mz = (absz&0x3FF) | 0x400, mz =
             0x400;
03389
                                  mx *= mx:
03390
                                 my *= my;
mz *= mz;
03391
03392
                                  int ix = mx \gg 21, iy = my \gg 21, iz = mz \gg 21;
                                  expx = 2*(expx+(absx**10)) - 15 + ix;
expy = 2*(expy+(absy**10)) - 15 + iy;
03393
03394
                                  \exp z = 2*(\exp z + (absz*10)) - 15 + iz;
03395
                                  mx «= 10 - ix;
03396
                                  my «= 10 - iy;
03397
                                  mz «= 10 - iz;
03398
03399
                                  int d = expy - expz;
03400
                                  mz = (d<30) ? ((mz&((static_cast<detail::uint32>(1) &d) -1))!=0)) : 1;
                                  my += mz;
03401
                                   if(my & 0x80000000)
03402
03403
                                   {
03404
                                           my = (my *1) | (my &1);
03405
                                            if(++expy > expx)
03406
                                                      std::swap(mx, my);
03407
03408
                                                      std::swap(expx, expy);
03409
                                            }
```

```
03410
              d = expx - expy;
03411
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,
      detail::float2half<half::round_style>(std::pow(detail::half2float<detail::internal_t>(x.data_),
      detail::half2float<detail::internal_t>(y.data_))));
03431
          #else
03432
              int absx = x.data_ \& 0x7FFF, absy = y.data_ \& 0x7FFF, exp = -15;
03433
              if(!absy || x.data_ == 0x3C00)
                  return half(detail::binary, detail::select(0x3C00, (x.data_==0x3C00) ? y.data_ :
03434
      x.data ));
03435
             bool is_int = absy >= 0x6400 \mid \mid (absy>=0x3C00 \&\& !(absy&((1*(25-(absy*10)))-1)));
03436
              unsigned int sign = x.data_ &
      (static_cast<unsigned>((absy<0x6800)&&is_int&&((absy»(25-(absy»10)))&1)) &1));
03437
              if(absx >= 0x7C00 \mid \mid absy >= 0x7C00)
                                                (absx>0x7C00 || absy>0x7C00) ? detail::signal(x.data_,
03438
                  return half(detail::binary,
      y.data_) :
03439
                                                (absy==0x7C00) ? ((absx==0x3C00) ? 0x3C00 : (!absx &&
      y.data_==0xFC00) ? detail::pole() :
03440
                                                (0x7C00&-((y.data_*)15)^(absx>0x3C00)))):
      (sign|(0x7C00&((y.data_>15)-1U))));
03441
              if(!absx)
                  return half(detail::binary, (y.data_&0x8000) ? detail::pole(sign) : sign);
03442
03443
              if((x.data_&0x8000) && !is_int)
03444
                  return half(detail::binary, detail::invalid());
03445
              if(x.data_ == 0xBC00)
03446
                  return half(detail::binary, sign|0x3C00);
03447
              switch(y.data_)
03448
03449
                  case 0x3800: return sqrt(x);
                  case 0x3C00: return half(detail::binary, detail::check_underflow(x.data_));
03450
03451
                  case 0x4000: return x * x;
03452
                  case 0xBC00: return half(detail::binary, 0x3C00) / x;
03453
              for(; absx<0x400; absx«=1,--exp) ;
detail::uint32 ilog = exp + (absx»10), msign = detail::sign_mask(ilog), f, m =</pre>
03454
03455
03456
      03/157
              for (exp=-11; m<0x80000000; m«=1,--exp);</pre>
03458
              for(; absy<0x400; absy«=1,--exp);</pre>
03459
              \texttt{m} = \texttt{detail::multiply64(m, static\_cast<detail::uint32>((absy&0x3FF)|0x400) & 21);}
              int i = m \gg 31;
03460
              exp += (absy *10) + i;
03461
              m «= 1 - i;
03462
03463
              if(exp < 0)
03464
              {
03465
                  f = m \gg -exp;
03466
                  exp = 0;
03467
03468
              else
03469
              {
03470
                  f = (m \ll exp) \& 0x7FFFFFFF;
03471
                  exp = m \gg (31-exp);
03472
              return half(detail::binary, detail::exp2_post<half::round_style>(f, exp,
03473
      ((msign&1)^(y.data_>15))!=0, sign));
03474
          #endif
03475
03476
03481
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)));
              *cos = half(detail::binary, detail::float2half<half::round_style>(std::cos(f)));
03496
03497
          #else
03498
              int abs = arg.data_ & 0x7FFF, sign = arg.data_ » 15, k;
              if(abs >= 0x7C00)
03499
                  *sin = *cos = half(detail::binary, (abs==0x7C00) ? detail::invalid() :
03500
      detail::signal(arg.data_));
03501
              else if(!abs)
03502
              {
03503
                   *sin = arg;
03504
                  *cos = half(detail::binary, 0x3C00);
03505
03506
              else if (abs < 0x2500)
03507
                  \star \texttt{sin} = \texttt{half(detail::binary, detail::rounded<half::round\_style,true>(arg.data\_-1, 1, 1));}
03508
03509
                  *cos = half(detail::binary, detail::rounded<half::round style,true>(0x3BFF, 1, 1));
```

```
}
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,
     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
03521
                     case 0x598C:
03522
                         *sin = half(detail::binary,
     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
03525
                     case 0x6A64:
                         *sin = half(detail::binary,
03526
     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;
                     case 0x6D8C:
03529
03530
                         *sin = half(detail::binary,
     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.second));
03544
03545
          #endif
03546
          }
03547
03556
          inline half sin(half arg)
03558
          #ifdef HALF_ARITHMETIC_TYPE
03559
             return half(detail::binary,
     detail::float2half<half::round_style>(std::sin(detail::half2float<detail::internal_t>(arg.data_))));
03560
          #else
03561
             int abs = arg.data & 0x7FFF, k;
03562
             if(!abs)
03563
                 return arg:
03564
             if(abs >= 0x7C00)
03565
                 return half(detail::binary, (abs==0x7C00) ? detail::invalid() :
     detail::signal(arg.data));
03566
             if(abs < 0x2900)
03567
                 return half(detail::binary, detail::rounded<half::round_style,true>(arg.data_-1, 1, 1));
03568
              if(half::round_style != std::round_to_nearest)
03569
                 switch(abs)
03570
03571
                      case 0x48B7: return half(detail::binary,
     detail::rounded<half::round_style,true>((~arg.data_&0x8000)|0x1D07, 1, 1));
                     case 0x6A64: return half(detail::binary,
     detail::rounded<half::round_style,true>((~arg.data_&0x8000)|0x3BFE, 1, 1));
03573
                     case 0x6D8C: return half(detail::binary,
     detail::rounded<half::round_style,true>((arg.data_&0x8000)|0x0FE6, 1, 1));
03574
              std::pair<detail::uint32,detail::uint32> sc = detail::sincos(detail::angle_arg(abs, k), 28);
03575
03576
             detail::uint32 sign = -static_cast<detail::uint32>(((k»1)&1)^(arg.data_»15));
             return half(detail::binary, detail::fixed2half<half::round_style,30,true,true,true>((((k&1) ?
     sc.second : sc.first)^sign) - sign));
03578
          #endif
03579
03580
         inline half cos(half arg)
03589
03590
03591
          #ifdef HALF_ARITHMETIC_TYPE
             return half(detail::binary,
03592
     detail::float2half<half::round_style>(std::cos(detail::half2float<detail::internal_t>(arg.data_))));
03593
         #else
03594
             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)
                   return half(detail::binary, detail::rounded<half::round_style,true>(0x3BFF, 1, 1));
03600
              if(half::round_style != std::round_to_nearest && abs == 0x598C)
03601
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);
              detail::uint32 sign = -static_cast<detail::uint32>(((k»1)^k)&1);
03604
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
           inline half tan(half arg)
03617
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)
                  return half(detail::binary, (abs==0x7C00) ? detail::invalid() :
03626
      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);
              detail::uint32 signy = detail::sign_mask(sc.first), signx = detail::sign_mask(sc.second);
detail::uint32 my = (sc.first^signy) - signy, mx = (sc.second^signx) - signx;
03638
03639
03640
               for(; my<0x80000000; my«=1,--exp);
              for(; mx<0x80000000; mx«=1,++exp);</pre>
03641
               return half(detail::binary, detail::tangent_post<half::round_style>(my, mx, exp,
03642
      (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)
                  return half(detail::binary, (abs>0x7C00) ? detail::signal(arg.data_) : (abs>0x3C00) ?
03663
      detail::invalid() :
03664
                                                detail::rounded<half::round_style,true>(sign|0x3E48, 0, 1));
03665
03666
                   return half(detail::binary, detail::rounded<half::round_style,true>(arg.data_, 0, 1));
03667
              if(half::round_style != std::round_to_nearest && (abs == 0x2B44 || abs == 0x2DC3))
              return half(detail::binary, detail::rounded<half::round_style,true>(arg.data_+1, 1, 1));
std::pair<detail::uint32,detail::uint32> sc = detail::atan2_args(abs);
03668
03669
              detail::uint32 m = detail::atan2(sc.first, sc.second,
03670
      (half::round_style==std::round_to_nearest) ? 27 : 26);
03671
              return half(detail::binary, detail::fixed2half<half::round_style,30,false,true,true>(m, 14,
      sign));
03672
           #endif
03673
03674
03683
           inline half acos(half arg)
03684
03685
           #ifdef HALF_ARITHMETIC_TYPE
              return half(detail::binary
03686
      detail::float2half<half::round_style>(std::acos(detail::half2float<detail::internal_t>(arg.data_))));
03687
          #else
03688
              unsigned int abs = arg.data_ & 0x7FFF, sign = arg.data_ » 15;
03689
03690
                   return half(detail::binary, detail::rounded<half::round_style,true>(0x3E48, 0, 1));
03691
               if(abs >= 0x3C00)
                  return half(detail::binary, (abs>0x7C00) ? detail::signal(arg.data_) : (abs>0x3C00) ?
03692
      detail::invalid() :
```

```
03693
                                                                        sign ? detail::rounded<half::round_style,true>(0x4248, 0, 1) :
         0);
                      std::pair<detail::uint32, detail::uint32> cs = detail::atan2_args(abs);
03694
03695
                      detail::uint32 m = detail::atan2(cs.second, cs.first, 28);
                      return half(detail::binary, detail::fixed2half<half::round_style,31,false,true,true>(sign ?
03696
         (0xC90FDAA2-m) : m, 15, 0, sign));
03697
                #endif
03698
03699
03708
               inline half atan(half arg)
03709
03710
               #ifdef HALF ARITHMETIC TYPE
03711
                      return half (detail::binary,
         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)
                           return half(detail::binary, (abs==0x7C00) ?
         detail::rounded<half::round_style,true>(sign|0x3E48, 0, 1) : detail::signal(arg.data_));
03718
                     if(abs <= 0x2700)
                      return half(detail::binary, detail::rounded<half::round_style,true>(arg.data_-1, 1, 1));
int exp = (abs>10) + (abs<=0x3FF);</pre>
03719
03720
                      detail::uint32 my = (absx0x3FF) | ((abs>0x3FF) x10);
detail::uint32 m = (exp>15) ? detail::atan2(myx19, 0x20000000)x(exp-15),
03721
03722
         (half::round_style==std::round_to_nearest) ? 26 : 24)
03723
                                                                        detail::atan2(my«(exp+4), 0x20000000,
         (half::round_style==std::round_to_nearest) ? 30 : 28);
                     return half(detail::binary, detail::fixed2half<half::round_style,30,false,true,true>(m, 14,
03724
         sign));
03725
                #endif
03726
03727
03738
               inline half atan2(half y, half x)
03739
03740
                #ifdef HALF ARITHMETIC TYPE
03741
                     return half (detail::binary,
         detail::float?half<half::round_style>(std::atan2(detail::half2float<detail::internal_t>(y.data_),
         detail::half2float<detail::internal_t>(x.data_))));
03742
                #else
03743
                     unsigned int absx = x.data_ & 0x7FFF, absy = y.data_ & 0x7FFF, signx = x.data_ » 15, signy =
         y.data_ & 0x8000;
03744
                      if(absx >= 0x7C00 || absy >= 0x7C00)
03745
                      {
                             if(absx > 0x7C00 || absy > 0x7C00)
03746
03747
                                   return half(detail::binary, detail::signal(x.data_, y.data_));
03748
                             if(absy == 0x7C00)
                                   return half(detail::binary, (absx<0x7C00) ?
03749
         detail::rounded<half::round_style,true>(signy|0x3E48, 0, 1) :
03750
                                                                                            signx ?
         detail::rounded<half::round_style,true>(signy|0x40B6, 0, 1) :
03751
         detail::rounded<half::round_style,true>(signy|0x3A48, 0, 1));
03752
                            return (x.data_==0x7C00) ? half(detail::binary, signy) : half(detail::binary,
         detail::rounded<half::round_style,true>(signy|0x4248, 0, 1));
03753
03754
                      if(!absv)
                           return signx ? half(detail::binary, detail::rounded<half::round_style,true>(signy|0x4248,
03755
         0, 1)) : y;
03756
                      if(!absx)
03757
                           return half(detail::binary, detail::rounded<half::round_style,true>(signy|0x3E48, 0, 1));
03758
                      int d = (absy*10) + (absy<=0x3FF) - (absx*10) - (absx<=0x3FF);
                      if(d > (signx ? 18 : 12))
03759
03760
                            return half(detail::binary, detail::rounded<half::round_style,true>(signy|0x3E48, 0, 1));
03761
                      if(signx && d < -11)</pre>
                            return half(detail::binary, detail::rounded<half::round_style,true>(signy|0x4248, 0, 1));
03762
03763
                      if(!signx && d < ((half::round_style==std::round_toward_zero) ? -15 : -9))</pre>
03764
                      {
03765
                             for(; absy<0x400; absy«=1,--d);</pre>
03766
                            detail::uint32 mx = ((absx«1)&0x7FF) | 0x800, my = ((absy«1)&0x7FF) | 0x800;
03767
                            int i = my < mx;
03768
                            d -= i;
                            if(d < -25)
03769
03770
                                  return half(detail::binary, detail::underflow<half::round_style>(signy));
03771
                            my \ll 11 + i;
                             return half (detail::binary,
03772
         detail::fixed2half<half::round_style,il,false,false,true>(my/mx, d+14, signy, my%mx!=0));
03773
03774
                      detail::uint32 m = detail::atan2(
                                                                                ((absv_{0}x_{3}FF)) | ((absv_{0}x_{3}FF) | ((absv
         0:-1)),
03775
                                                                                ((absx&0x3FF)|((absx>0x3FF)&10))&(19-((d>0) ? d : (d<0) ?
         0:1));
03776
                      return half(detail::binary, detail::fixed2half<half::round_style,31,false,true,true>(signx ?
         (0xC90FDAA2-m) : m, 15, signy, signx));
03777
                #endif
03778
                }
```

7.5 half.hpp 237

```
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
03798
               int abs = arg.data_ & 0x7FFF, exp;
03799
               if(!abs || abs >= 0x7C00)
                   return (abs>0x7C00) ? half(detail::binary, detail::signal(arg.data)) : arg;
03800
03801
               if(abs <= 0x2900)
               return half(detail::binary, detail::rounded<half::round_style,true>(arg.data_, 0, 1));
std::pair<detail::uint32,detail::uint32> mm = detail::hyperbolic_args(abs, exp,
03802
      (half::round_style==std::round_to_nearest) ? 29 : 27);
03804
               detail::uint32 m = mm.first - mm.second;
               for(exp+=13; m<0x80000000 && exp; m<=1,--exp);</pre>
03805
03806
               unsigned int sign = arg.data_ & 0x8000;
03807
               if(exp > 29)
03808
                   return half(detail::binary, detail::overflow<half::round_style>(sign));
               return half (detail::binary, detail::fixed2half<half::round_style, 31, false, false, true>(m, exp,
03809
      sign));
03810
           #endif
03811
           }
03812
03821
           inline half cosh(half arg)
03822
03833
           #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)
               return half(detail::binary, (abs>0x7C00) ? detail::signal(arg.data_) : 0x7C00);
std::pair<detail::uint32, detail::uint32> mm = detail::hyperbolic_args(abs, exp,
03830
03831
       (half::round_style==std::round_to_nearest) ? 23 : 26);
03832
               detail::uint32 m = mm.first + mm.second, i = (~m&0xFFFFFFFF) » 31;
03833
               m = (m \cdot i) \mid (m \cdot i) \mid 0 \times 80000000;
03834
               if((exp+=13+i) > 29)
                   return half(detail::binary, detail::overflow<half::round_style>());
03835
               return half(detail::binary, detail::fixed2half<half::round_style,31,false,false,true>(m,
03836
      exp));
03837
           #endif
03838
03839
03848
           inline half tanh (half arg)
03849
           #ifdef HALF_ARITHMETIC_TYPE
03850
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)
                   return half(detail::binary, detail::rounded<half::round_style,true>(arg.data_-3, 0, 1));
03863
03864
               std::pair<detail::uint32,detail::uint32> mm = detail::hyperbolic_args(abs, exp, 27);
      detail::uint32 my = mm.first - mm.second - (half::round_style!=std::round_to_nearest), mx =
mm.first + mm.second, i = (~mx&0xFFFFFFFF) » 31;
03865
03866
               for (exp=13; my<0x80000000; my«=1,--exp);</pre>
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
           #if defined(HALF ARITHMETIC TYPE) && HALF ENABLE CPP11 CMATH
03882
               return half(detail::binary,
03883
      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 \mid | 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));
```

```
if(half::round_style != std::round_to_nearest)
03891
                  switch (abs)
03892
                      case 0x32D4: return half(detail::binary,
03893
      detail::rounded<half::round_style,true>(arg.data_-13, 1, 1));
03894
                      case 0x3B5B: return half(detail::binary,
      detail::rounded<half::round_style,true>(arg.data_-197, 1, 1));
03895
03896
              return half(detail::binary, detail::area<half::round_style,true>(arg.data_));
03897
          #endif
03898
          }
03899
03908
          inline half acosh (half arg)
03909
03910
          #if defined(HALF_ARITHMETIC_TYPE) && HALF_ENABLE_CPP11_CMATH
03911
              return half(detail::binary,
      detail::float2half<half::round_style>(std::acosh(detail::half2float<detail::internal_t>(arg.data_))));
03912
          #else
03913
             int abs = arg.data_ & 0x7FFF;
03914
              if((arg.data_&0x8000) || abs < 0x3C00)</pre>
                  return half(detail::binary, (abs<=0x7C00) ? detail::invalid() :</pre>
03915
     detail::signal(arg.data_));
03916
             if(abs == 0x3C00)
                  return half(detail::binary, 0);
03917
03918
              if(arg.data_ >= 0x7C00)
                  return (abs>0x7C00) ? half(detail::binary, detail::signal(arg.data_)) : arg;
03919
03920
              return half(detail::binary, detail::area<half::round_style,false>(arg.data_));
03921
          #endif
03922
03923
03933
          inline half atanh (half arg)
03934
03935
          #if defined(HALF_ARITHMETIC_TYPE) && HALF_ENABLE_CPP11_CMATH
              return half(detail::binary,
03936
     detail::float2half<half::round_style>(std::atanh(detail::half2float<detail::internal_t>(arg.data_))));
03937
          #else
03938
              int abs = arg.data_ & 0x7FFF, exp = 0;
03939
              if(!abs)
03940
                  return arg;
03941
              if(abs >= 0x3C00)
03942
                  return half(detail::binary, (abs==0x3C00) ? detail::pole(arg.data_&0x8000) : (abs<=0x7C00)</pre>
     ? detail::invalid() : detail::signal(arg.data_));
03943
             if(abs < 0x2700)
03944
                  return half(detail::binary, detail::rounded<half::round_style,true>(arg.data_, 0, 1));
              detail::uint32 m = static_cast<detail::uint32>((abs&0x3FF)|((abs>0x3FF) w10)) w
03945
      ((abs>10)+(abs<=0x3FF)+6), my = 0x80000000 + m, mx = 0x80000000 - m;
03946
              for(; mx<0x80000000; mx«=1,++exp);</pre>
              int i = my >= mx, s;
return half(detail::binary, detail::log2_post<half::round_style,0xB8AA3B2A>(detail::log2(
03947
03948
                  (detail::divide64(my»i, mx, s)+1)»1, 27)+0x10, exp+i-1, 16, arg.data_&0x8000));
03949
03950
          #endif
03951
03952
03957
          inline half erf(half arg)
03966
03967
03968
          #if defined(HALF ARITHMETIC TYPE) && HALF ENABLE CPP11 CMATH
              return half(detail::binary,
03969
      detail::float2half<half::round_style>(std::erf(detail::half2float<detail::internal_t>(arg.data_))));
03970
          #else
03971
              unsigned int abs = arg.data_ & 0x7FFF;
              if(!abs | | abs \rangle = 0x7C00)
03972
03973
                  return (abs>=0x7C00) ? half(detail::binary, (abs==0x7C00) ? (arg.data_-0x4000) :
      detail::signal(arg.data_)) : arg;
03974
              if(abs >= 0x4200)
                  return half(detail::binary,
03975
     detail::rounded<half::round_style,true>((arg.data_&0x8000)|0x3BFF, 1, 1));
              return half(detail::binary, detail::erf<half::round_style,false>(arg.data_));
03976
03977
          #endif
03978
03979
03988
          inline half erfc(half arg)
03989
          #if defined(HALF ARITHMETIC TYPE) && HALF ENABLE CPP11 CMATH
03990
              return half (detail::binary,
03991
      detail::float2half<half::round_style>(std::erfc(detail::half2float<detail::internal_t>(arg.data_))));
03992
03993
             unsigned int abs = arg.data_ & 0x7FFF, sign = arg.data_ & 0x8000;
              if(abs >= 0x7C00)
03994
                  return (abs>=0x7C00) ? half(detail::binary, (abs==0x7C00) ? (sign»1) :
03995
     detail::signal(arg.data_)) : arg;
03996
              if(!abs)
03997
                  return half(detail::binary, 0x3C00);
03998
              if(abs >= 0x4400)
03999
                  return half(detail::binary, detail::rounded<half::round_style,true>((sign*1)-(sign*15),
     sign»15, 1));
04000
              return half (detail::binary, detail::erf<half::round style,true>(arg.data));
```

7.5 half.hpp 239

```
#endif
04001
04002
04003
04013
           inline half lgamma(half arg)
04014
04015
           #if defined(HALF_ARITHMETIC_TYPE) && HALF_ENABLE_CPP11_CMATH
04016
               return half (detail::binary,
      detail::float2half<half::round_style>(std::lgamma(detail::half2float<detail::internal_t>(arg.data_))));
04017
           #else
04018
               int abs = arg.data_ & 0x7FFF;
               if(abs >= 0x7C00)
04019
               return half(detail::binary, (abs==0x7C00) ? 0x7C00 : detail::signal(arg.data_));
if(!abs || arg.data_ >= 0xE400 || (arg.data_ >= 0xBC00 && !(abs&((1«(25-(abs»10)))-1))))
    return half(detail::binary, detail::pole());
04020
04021
04022
04023
               if(arg.data\_ == 0x3C00 \mid \mid arg.data\_ == 0x4000)
                    return half(detail::binary, 0);
04024
04025
               return half(detail::binary, detail::gamma<half::round_style,true>(arg.data_));
04026
           #endif
04027
04028
04038
           inline half tgamma(half arg)
04039
04040
           #if defined(HALF ARITHMETIC TYPE) && HALF ENABLE CPP11 CMATH
               return half(detail::binary
04041
      detail::float2half<half::round_style>(std::tgamma(detail::half2float<detail::internal_t>(arg.data_))));
04042
           #else
04043
               unsigned int abs = arg.data_ & 0x7FFF;
04044
               if(!abs)
04045
                    return half(detail::binary, detail::pole(arg.data_));
04046
               if(abs >= 0x7C00)
04047
                   return (arg.data_==0x7C00) ? arg : half(detail::binary, detail::signal(arg.data_));
               if(arg.data_ >= 0xE400 || (arg.data_ >= 0xE000 && !(abs&((1*(25-(abs*10)))-1))))
    return half(detail::binary, detail::invalid());
04048
04049
04050
               if(arg.data_ >= 0xCA80)
      return half(detail::binary,
detail::underflow<half::round_style>((1-((abs»(25-(abs»10)))&1))%1));
04051
               if(arg.data_ <= 0x100 || (arg.data_ >= 0x4000 && arg.data_ < 0x8000))
    return half(detail::binary, detail::overflow<half::round_style>());
04052
04053
04054
               if(arg.data_ == 0x3C00)
04055
                   return arg;
04056
               return half(detail::binary, detail::gamma<half::round_style,false>(arg.data_));
04057
           #endif
04058
04059
04064
04071
           inline half ceil(half arg) { return half(detail::binary,
      detail::integral<std::round_toward_infinity,true,true>(arg.data_)); }
04072
04079
           inline half floor(half arg) { return half(detail::binary,
      detail::integral<std::round_toward_neg_infinity,true,true>(arg.data_)); }
04080
04087
           inline half trunc(half arg) { return half(detail::binary,
      detail::integral<std::round_toward_zero,true,true>(arg.data_)); }
04088
           inline half round(half arg) { return half(detail::binary,
04095
      detail::integral<std::round to nearest,false,true>(arg.data )); }
04096
04102
           inline long lround(half arg) { return
      detail::half2int<std::round_to_nearest, false, false, long>(arg.data_); }
04103
04110
           inline half rint(half arg) { return half(detail::binary,
      detail::integral<half::round_style,true,true>(arg.data_)); }
04111
04118
           inline long lrint(half arg) { return
      detail::half2int<half::round_style,true,true,long>(arg.data_); }
04119
04125
           inline half nearbyint (half arg) { return half (detail::binary,
      detail::integral<half::round_style,true,false>(arg.data_)); }
04126 #if HALF_ENABLE_CPP11_LONG_LONG
04132
           inline long long llround(half arg) { return
      detail::half2int<std::round_to_nearest, false, false, long long>(arg.data_); }
04133
04140
           inline long long llrint(half arg) { return detail::half2int<half::round_style,true,true,long</pre>
      long>(arg.data_); }
04141 #endif
04142
04147
04154
           inline half frexp(half arg, int *exp)
04155
04156
               \star exp = 0:
               unsigned int abs = arg.data_ & 0x7FFF;
04157
04158
               if(abs >= 0x7C00 || !abs)
                    return (abs>0x7C00) ? half(detail::binary, detail::signal(arg.data_)) : arg;
04159
04160
               for(; abs<0x400; abs<=1,--*exp);</pre>
               *exp += (abs > 10) - 14;
04161
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;
               if(abs \ge 0x7C00 || !abs)
04177
04178
                   return (abs>0x7C00) ? half(detail::binary, detail::signal(arg.data_)) : arg;
04179
               for(; abs<0x400; abs«=1,--exp);</pre>
04180
               exp += abs > 10;
04181
               if(exp > 30)
04182
                   return half(detail::binary, detail::overflow<half::round_style>(sign));
04183
               else if (exp < -10)
                   return half(detail::binary, detail::underflow<half::round_style>(sign));
04184
04185
               else if (exp > 0)
               return half(detail::binary, sign|(exp<10)|(abs&0x3FF));
unsigned int m = (abs&0x3FF) | 0x400;</pre>
04186
04187
04188
               return half(detail::binary, detail::rounded<half::round_style,false>(sign|(m»(1-exp))),
      (m \rightarrow -exp) \& 1, (m\& ((1 \leftarrow -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
           inline half modf(half arg, half *iptr)
04219
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)
               return iptr->data_ = arg.data_ & 0x8000, arg;
unsigned int exp = abs » 10, mask = (1«(25-exp)) - 1, m = arg.data_ & mask;
04230
04231
04232
               iptr->data_ = arg.data_ & ~mask;
               <u>if</u>(!m)
04233
04234
                    return half(detail::binary, arg.data_&0x8000);
04235
               for(; m<0x400; m«=1,--exp);</pre>
04236
               return half(detail::binary, (arg.data_&0x8000)|(exp<10)|(m&0x3FF));</pre>
04237
          }
04238
04247
           inline int ilogb(half arg)
04248
          {
04249
               int abs = arg.data_ & 0x7FFF, exp;
04250
               if(!abs || abs >= 0x7C00)
04251
               {
                   detail::raise(FE INVALID);
04252
04253
                   return !abs ? FP_ILOGBO : (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 \ge 0 \times 7C00)
               return half(detail::binary, (abs==0x7C00) ? 0x7C00 : detail::signal(arg.data_));
for(exp=(abs>10)-15; abs<0x200; abs<=1,--exp);
unsigned int value = static_cast<unsigned>(exp<0) < 15;</pre>
04271
04272
04273
04274
               if(exp)
04275
               {
04276
                   unsigned int m = std::abs(exp) « 6;
04277
                   for (exp=18; m<0x400; m<=1,--exp);
04278
                   value |= (exp \ll 10) + m;
04279
04280
               return half(detail::binary, value);
04281
           }
04282
           inline half nextafter (half from, half to)
04291
04292
               int fabs = from.data_ & 0x7FFF, tabs = to.data_ & 0x7FFF;
04293
04294
               if(fabs > 0x7C00 || tabs > 0x7C00)
04295
                    return half(detail::binary, detail::signal(from.data_, to.data_));
               if(from.data_ == to.data_ || !(fabs|tabs))
04296
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
```

7.5 half.hpp 241

```
(from.data_^(0x8000|(0x8000-(from.data_»15))))<(to.data_^(0x8000|(0x8000-(to.data_»15))))))))) - 1;
04305
              detail::raise(FE_OVERFLOW, fabs<0x7C00 && (out&0x7C00) ==0x7C00);</pre>
04306
              detail::raise(FE_UNDERFLOW, !HALF_ERRHANDLING_UNDERFLOW_TO_INEXACT && (out&0x7C00)<0x400);
04307
              return half(detail::binary, out);
04308
          }
04309
04318
          inline half nexttoward(half from, long double to)
04319
04320
              int fabs = from.data_ & 0x7FFF;
              if(fabs > 0x7C00)
04321
                  return half(detail::binary, detail::signal(from.data_));
04322
              long double lfrom = static_cast<long double>(from);
04323
04324
              if (detail::builtin_isnan(to) || lfrom == to)
04325
                  return half(static_cast<float>(to));
04326
              if(!fabs)
04327
                  detail::raise(FE_UNDERFLOW, !HALF_ERRHANDLING_UNDERFLOW_TO_INEXACT);
04328
04329
                  return half(detail::binary, (static_cast<unsigned>(detail::builtin_signbit(to)) <15) +1);</pre>
04330
04331
              unsigned int out = from.data_ + (((from.data_**15)^static_cast<unsigned>(lfrom<to))*(1) - 1;</pre>
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)); }
0/3/3
04348
04357
          inline HALF CONSTEXPR int fpclassifv(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
          inline HALF CONSTEXPR bool isnam(half arg) { return (arg.data_&0x7FFF) > 0x7C00; }
04385
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
          {
              return ((x.data_^(0x8000|(0x8000-(x.data_*)15))))+(x.data_*)15))) >=
04425
      ((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)) <</pre>
      ((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 v)
          {
04447
              return ((x.data_^(0x8000|(0x8000-(x.data_»15))))+(x.data_»15)) <=</pre>
      ((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
04473
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
```

```
04518
           inline int feclearexcept(int excepts) { detail::errflags() &= ~excepts; return 0; }
04519
          inline int fetestexcept(int excepts) { return detail::errflags() & excepts; }
04527
04528
          inline int feraiseexcept(int excepts) { detail::errflags() |= excepts; detail::raise(excepts);
04538
      return 0; }
04539
04548
           inline int fegetexceptflag(int *flagp, int excepts) { *flagp = detail::errflags() & excepts;
      return 0; }
04549
           inline int fesetexceptflag(const int *flagp, int excepts) { detail::errflags() =
04559
      (detail::errflags()|(*flagp&excepts)) & (*flagp|~excepts); return 0; }
04560
04572
           inline void fethrowexcept(int excepts, const char *msg = "")
04573
               excepts &= detail::errflags();
04574
04575
               if(excepts & (FE_INVALID|FE_DIVBYZERO))
                   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
       #pragma warning(pop)
#undef HALF_POP_WARNINGS
04597
04598
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
          public:
00053
              Light(const glm::vec3& pos, const glm::vec3& col, float ambient, float diffuse);
00065
00066
08000
              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 243

7.7 Mesh.hpp

```
00001 //
00002 //
          Mesh.hpp
00003 //
          GL_Geometry
00004 //
          Created by Alonso García on 11/12/24.
00005 //
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
00039
          class Mesh
00040
          private:
00041
00045
               enum
00046
               {
00047
                   COORDINATES_VBO,
                   COLORS_VBO,
NORMALS_VBO,
00048
00049
                   INDEXES_VBO,
TEXTURE_UV_VBO,
00050
00051
                   VBO_COUNT
00053
               } ;
00054
00058
               MeshType m_mesh_type;
00059
00060
          protected:
00064
              std::vector<glm::vec3> coordinates;
00065
               std::vector<glm::vec3> colors;
00066
               std::vector<glm::vec3> normals;
00067
               std::vector<GLuint> indices;
00068
               std::vector<glm::vec2> texture_uvs;
00069
00073
               GLsizei number_of_vertices;
00074
00080
               void create_mesh(std::string mesh_name = "");
00081
00082
          private:
               GLuint vbo_ids[VBO_COUNT];
GLuint vao_id;
00086
00087
00088
00092
               glm::mat4 model_view_matrix;
00093
               glm::mat4 normal_matrix;
00094
00098
               std::shared_ptr < udit::Shader > m_shader;
00099
00100
          public:
00104
              Mesh();
00105
00114
               Mesh(std::string & path);
00115
00126
               static std::shared_ptr <Mesh> make_mesh(MeshType type, const std::string &path = "");
00127
00133
00134
00142
               virtual void translate(glm::vec3 translation);
00143
00152
               virtual void rotate(glm::vec3 rotation, float angle);
00153
00161
               virtual void scale(glm::vec3 scale);
00162
00168
               virtual void update();
00169
00177
               virtual void render(glm::mat4 view matrix);
00178
               virtual void resize(glm::mat4 projection_matrix);
00187
```

```
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
00010 #include "glad.h"
00011
00012 #include "Mesh.hpp"
00013
00014 namespace udit
00015 {
00025
          class Plane : public Mesh
00026
          private:
00027
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.

7.10 Scene.hpp 245

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
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
           private:
00036
00043
                std::vector<std::string> skybox faces =
00044
                        "skybox_east.jpg", "skybox_west.jpg", "skybox_up.jpg", "skybox_down.jpg", "skybox_north.jpg", "skybox_south.jpg"
00045
00046
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
                unsigned width, height;
00068
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 <u>Shader</u> 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 247

7.12 Shader.hpp

Go to the documentation of this file.

```
00001 //
00002 //
          Shader.hpp
00003 //
          GL_Geometry
00004 //
          Created by Alonso García on 11/12/24.
00005 //
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 =
      \hbox{"/Users/alonsoggdev/UDIT/Asignaturas/Programacion\_Grafica/GL\_Scene/GL\_Scene/";}
00063
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;</pre>
00087
          public:
00088
00095
              Shader();
00096
              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(
                  udit::ShaderType type = udit::ShaderType::DEFAULT,
const std::string & vertex_shader = "",
00126
00127
                  const std::string & fragment_shader = "",
00128
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;
```

```
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 //
          Created by Alonso García on 21/12/24.
00005 //
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 =
     "/Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/resources/skybox/";
00040
00041
         public:
00047
             Skybox();
00048
             Skybox(float size, const std::vector<std::string>& faces);
00056
00062
             unsigned int getCubemapTexture() const { return cubemapTexture; }
00063
         private:
00064
00074
             void loadCubemap(const std::vector<std::string>& faces);
00075
00076 }
```

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
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>
```

7.17 Window.hpp 251

```
00011 #include <string>
00012 #include <utility>
00013
00014 namespace udit
00015 {
00016
          class Window
00018
00019
          public:
00020
00021
               enum Position
00022
               {
                   UNDEFINED = SDL_WINDOWPOS_UNDEFINED,
CENTERED = SDL_WINDOWPOS_CENTERED,
00023
00024
00025
00026
00027
               struct OpenGL_Context_Settings
00028
00029
                   unsigned version_major
                                                   = 3;
00030
                   unsigned version_minor
                                                  = 3;
00031
                            core_profile
                                                   = true;
                   unsigned depth_buffer_size = 24;
00032
00033
                   unsigned stencil_buffer_size = 0;
                                                  = true;
00034
                   bool
                           enable_vsync
00035
               };
00036
00037
          private:
00038
               SDL_Window * window_handle;
00039
00040
               SDL_GLContext opengl_context;
00041
00042
          public:
00043
00044
               Window
00045
                   const std::string & title,
00046
00047
                   int left_x,
int top_y,
00048
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
               Window
00058
00059
               (
00060
                   const char * title,
                   int left_x, int top_y,
00061
00062
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
               {
                   this->window_handle = std::exchange (other.window_handle, nullptr);
this->opengl_context = std::exchange (other.opengl_context, nullptr);
00078
00079
08000
               }
00081
00082
               Window & operator = (Window && other) noexcept
00083
               {
                   this->window_handle = std::exchange (other.window_handle, nullptr);
00084
00085
                   this->opengl_context = std::exchange (other.opengl_context, nullptr);
00086
00087
88000
               }
00089
00090
          public:
00091
00092
               void swap_buffers ();
00093
00094
          };
00095
00096 }
```

Index

\sim Mesh	Camera, 57
Mesh, 119	Camera, 58
udit::Mesh, 125	front, 59
\sim Shader	get_view_matrix, 58
Shader, 143	mouse_sensitivity, 59
udit::Shader, 149	movement_speed, 59
	pitch, 59
abs	position, 59
half_float, 12	process_keyboard, 58
acos	process_mouse_movement, 58
half_float, 13	right, 60
half_float::half, 77	up, 60
acosh	world_up, 60
half_float, 13	yaw, <mark>60</mark>
half_float::half, 77	zoom, 60
angle_arg	cbrt
half.hpp, 174	half float, 16
area	half_float::half, 80
half.hpp, 174	ceil
arithmetic_shift	half_float, 16
half.hpp, 175	half_float::half, 80
asin	check underflow
half_float, 14	half.hpp, 177
half_float::half, 78	COLOR
asinh	Texture.hpp, 250
half_float, 14	compile_shaders
half_float::half, 78	Shader, 144
atan	udit::Shader, 149
half_float, 14	
half_float::half, 78	compsignal
atan2	half.hpp, 178
half.hpp, 175	copysign
half_float, 15	half_float, 17
half_float::half, 79	half_float::half, 81
atan2 args	COS
	half_float, 17
half.hpp, 176	half_float::half, 81
atanh half float, 15	cosh
- :	half_float, 17
half_float::half, 79	half_float::half, 81
bind	create_mesh
	Mesh, 119
Texture, 159	udit::Mesh, 125
udit::Texture, 161	Cube, 61
builtin_isinf	udit::Cube, 63, 64
half.hpp, 176	
builtin_isnan	DEFAULT
half.hpp, 176	Shader.hpp, 246
builtin_signbit	divide64
half.hpp, 177	half.hpp, 178

a.uf	half flacturbalf OC
erf	half_float::half, 86
half.hpp, 178	fmin
half_float, 18	half_float; 25
half_float::half, 82 erfc	half_float::half, 86 fmod
	half float, 25
half_float, 18 half_float::half, 82	half_float::half, 87
errflags	FP FAST FMAH
half.hpp, 179	half.hpp, 173
EventHandler, 65	fpclassify
EventHandler, 65	half_float, 25
handle_events, 65	half_float::half, 87
exp	frexp
half_float, 19	half_float, 26
half_float::half, 83	half_float::half, 88
exp2	front
half.hpp, 179	Camera, 59
half_float, 19	
half_float::half, 83	gamma
exp2_post	half.hpp, 184
half.hpp, 179	GEOMETRY
expm1	Shader.hpp, 246
half_float, 20	get_model_view_matrix
half_float::half, 84	Mesh, 119
101	udit::Mesh, 125
f31	get_model_view_matrix_id
half_float::detail::f31, 66, 67 fabs	Shader, 144
half_float, 20	udit::Shader, 150
half_float::half, 84	get_name Shader, 144
fdim	udit::Shader, 150
half_float, 20	get_normal_matrix_id
half_float::half, 84	Shader, 144
feclearexcept	udit::Shader, 150
half float, 21	get_program_id
fegetexceptflag	Shader, 144
half_float, 21	udit::Shader, 150
feraiseexcept	get_projection_matrix_id
half_float, 22	Shader, 145
fesetexceptflag	udit::Shader, 150
half_float, 22	get_shader_matrix_ids
fetestexcept	Mesh, 120
half_float, 22	udit::Mesh, 125
fethrowexcept	get_shader_program_id
half_float, 23	Mesh, 120
fixed2half	udit::Mesh, 125
half.hpp, 181	get_view_matrix
float2half	Camera, 58
half.hpp, 182	getCubemapTexture
float2half_impl	Skybox, 155
half.hpp, 182, 183	udit::Skybox, 158
floor	GL_Scene/Camera.hpp, 165
half_float, 23	GL_Scene/Cube.hpp, 166
half_float::half, 85 fma	GL_Scene/EventHandler.hpp, 166
half_float, 24	GL_Scene/half.hpp, 166, 198 GL_Scene/Light.hpp, 242
half_float::half, 85	GL_Scene/Light.npp, 242 GL_Scene/Mesh.hpp, 243
fmax	GL_Scene/Plane.hpp, 244
half_float, 24	GL_Scene/Scene.hpp, 244, 245

GL_Scene/Shader.hpp, 246, 247	underflow, 197
GL_Scene/Skybox.hpp, 248	half2float
GL_Scene/Texture.hpp, 249, 250	half.hpp, 184
GL_Scene/Window.hpp, 250	half2float_impl
	half.hpp, 186
half	half2int
half_float::half, 71	half.hpp, 187
half.hpp	half_cast
angle_arg, 174	half_float, 26, 27
area, 174	HALF_ENABLE_F16C_INTRINSICS
arithmetic_shift, 175	half.hpp, 173
atan2, 175	HALF_ERRHANDLING_OVERFLOW_TO_INEXACT
atan2_args, 176	half.hpp, 173
builtin_isinf, 176	HALF_ERRHANDLING_UNDERFLOW_TO_INEXACT
builtin_isnan, 176	half.hpp, 173
builtin_signbit, 177	half_float, 9
check_underflow, 177	abs, 12
compsignal, 178	acos, 13
divide64, 178	acosh, 13
erf, 178	asin, 14
errflags, 179	asinh, 14
exp2, 179	atan, 14
exp2_post, 179	atan2, 15
fixed2half, 181	atanh, 15
float2half, 182	cbrt, 16
float2half_impl, 182, 183	ceil, 16
FP_FAST_FMAH, 173	copysign, 17
gamma, 184	cos, 17
half2float, 184	cosh, 17
half2float_impl, 186	erf, 18
half2int, 187	erfc, 18
HALF_ENABLE_F16C_INTRINSICS, 173	exp, 19
HALF_ERRHANDLING_OVERFLOW_TO_INEXAC	T, exp2, 19
173	expm1, 20
HALF_ERRHANDLING_UNDERFLOW_TO_INEXA	CT, fabs, 20
173	fdim, 20
HALF_ROUND_STYLE, 173	feclearexcept, 21
HLF_ROUNDS, 174	fegetexceptflag, 21
HUGE_VALH, 174	feraiseexcept, 22
hyperbolic_args, 187	fesetexceptflag, 22
hypot_post, 188	fetestexcept, 22
int2half, 188	fethrowexcept, 23
integral, 189	floor, 23
invalid, 189	fma, 24
log2, 190	fmax, 24
log2_post, 190	fmin, 25
mod, 191	fmod, 25
mulhi, 191	fpclassify, 25
multiply64, 191	frexp, 26
overflow, 192	half_cast, 26, 27
pole, 192	hypot, 28
raise, 193	ilogb, 29
rounded, 193	isfinite, 29
select, 194	isgreater, 29
sign_mask, 194	isgreaterequal, 30
signal, 194, 195	isinf, 30
sincos, 196	isless, 30
sqrt, 196	islessequal, 31
tangent_post, 196	·

islessgreater, 31	f31, 66, 67
isnan, 31	operator+, 67
isnormal, 32	operator-, 67
isunordered, 32	operator/, 68
ldexp, 32	operator*, 67
Igamma, 33	half_float::detail::half_caster< half, half, R >, 112
log, 33	half_float::detail::half_caster< half, U, R >, 112
log10, 34	half_float::detail::half_caster< T, half, R >, 112
log1p, 34	half float::detail::half caster< T, U, R >, 111
log2, 35	half_float::detail::is_float< const T >, 113
logb, 35	half_float::detail::is_float< const volatile T >, 114
Irint, 36	half_float::detail::is_float< double >, 114
Iround, 36	half_float::detail::is_float< float >, 114
modf, 36	half_float::detail::is_float< long double >, 115
nanh, 37	half_float::detail::is_float< typename >, 113
nearbyint, 37	half_float::detail::is_float< volatile T >, 115
nextafter, 37	half float::half, 68
	-
nexttoward, 38	acos, 77
operator!=, 38	acosh, 77
operator<, 41	asin, 78
operator<<, 42	asinh, 78
operator<=, 42	atan, 78
operator>, 43	atan2, 79
operator>>, 44	atanh, 79
operator>=, 43	cbrt, 80
operator+, 39	ceil, 80
operator-, 40	copysign, 81
operator/, 41	cos, 81
operator==, 42	cosh, 81
operator*, 39	erf, 82
pow, 44	erfc, 82
remainder, 45	exp, 83
remquo, 45	exp2, 83
rint, 46	expm1, 84
round, 46	fabs, 84
rsqrt, 46	fdim, 84
scalbin, 47	floor, 85
scalbn, 47	fma, 85
signbit, 48	fmax, 86
sin, 48	fmin, 86
sincos, 49	fmod, 87
sinh, 49	fpclassify, 87
sqrt, 50	frexp, 88
tan, 50	half, 71
tanh, 50	hypot, 88, 89
tgamma, 51	ilogb, 89
trunc, 51	isfinite, 90
half float::detail::binary t, 53	isgreater, 90
half_float::detail::bits< const T >, 54	isgreaterequal, 90
half_float::detail::bits< const volatile T >, 54	isinf, 91
half_float::detail::bits< double >, 55	isless, 91
half_float::detail::bits< float >, 55	islessequal, 91
half_float::detail::bits < T >, 53	islessgreater, 92
half_float::detail::bits< volatile T >, 56	isnan, 92
half_float::detail::bool_type< bool >, 56	isnormal, 92
half_float::detail::conditional< bool, T, typename >, 61	Igamma, 93
half_float::detail::conditional< false, T, F >, 61	log, 93
half_float::detail::f31, 66	log10, 94

log1p, 94	half.hpp, 174
log2, 95	hyperbolic_args
logb, 95	half.hpp, 187
Irint, 96	hypot
Iround, 96	half_float, 28
modf, 96	half_float::half, 88, 89
nanh, 97	hypot_post
nearbyint, 97	half.hpp, 188
nextafter, 97	
nexttoward, 98	ilogb
operator float, 71	half_float, 29
operator!=, 98	half_float::half, 89
operator<, 101	int2half
operator<<, 101	half.hpp, 188
operator<=, 102	integral
operator>, 103	half.hpp, 189
operator>>, 104	invalid
operator>=, 103	half.hpp, 189
operator+, 99	is_loaded
operator++, 72, 73	Texture, 159
operator+=, 73	udit::Texture, 161
operator-, 100	isfinite
operator, 74	half_float, 29
operator-=, 74, 75	half_float::half, 90
operator/, 100	isgreater
operator/=, 75, 76	half_float, 29
operator=, 76	half_float::half, 90
operator==, 102	isgreaterequal
operator*, 99	half_float, 30
operator*=, 71, 72	half_float::half, 90
pow, 104	isinf
remainder, 105	half_float, 30
remquo, 105	half_float::half, 91
rint, 106	isless
round, 106	half_float, 30
rsqrt, 106	half_float::half, 91
scalbln, 107	islessequal
signbit, 107	half_float, 31
sin, 108	half_float::half, 91
sincos, 108	islessgreater
sinh, 108	half_float, 31
sqrt, 109	half_float::half, 92
tan, 109	isnan
tanh, 110	half_float, 31
tgamma, 110	half_float::half, 92
trunc, 111	isnormal
HALF_ROUND_STYLE	half_float, 32
half.hpp, 173	half_float::half, 92
handle_events	isunordered
EventHandler, 65	half_float, 32
has_textures	
Shader, 145	ldexp
udit::Shader, 151	half_float, 32
HEIGHT	lgamma
Texture.hpp, 250	half_float, 33
HLF ROUNDS	half_float::half, 93
half.hpp, 174	Light
HUGE VALH	udit::Light, 116
	load_texture

	Texture, 159	Camera, 59
	udit::Texture, 162	movement_speed
log		Camera, 59
	half_float, 33	mulhi
	half_float::half, 93	half.hpp, 191
log1		multiply64
	half_float, 34	half.hpp, 191
	half_float::half, 94	nanh
log1		half_float, 37
	half_float, 34	half_float::half, 97
l:0	half_float::half, 94	nearbyint
log2		half float, 37
	half.hpp, 190	half_float::half, 97
	half_float, 35	nextafter
l==0	half_float::half, 95	half_float, 37
iog2	_post	half_float::half, 97
ماند ما	half.hpp, 190	nexttoward
logb		half float, 38
	half_float, 35	half_float::half, 98
المساسا	half_float::half, 95	nan_noatnan, 30
Irint	half float OC	operator float
	half_float, 36	half_float::half, 71
Irour	half_float::half, 96	operator!=
iroui		half_float, 38
	half_float; 36 half_float::half, 96	half_float::half, 98
	nan_noatnan, 90	operator<
mak	e_light	half_float, 41
	udit::Light, 116	half_float::half, 101
mak	e_mesh	operator<<
	Mesh, 120	half_float, 42
	udit::Mesh, 126	half_float::half, 101
mak	e_shader	operator<=
	Shader, 145	half_float, 42
	udit::Shader, 151	half_float::half, 102
Mes	h, 117	operator>
	∼Mesh, 119	half_float, 43
	create_mesh, 119	half_float::half, 103
	get_model_view_matrix, 119	operator>>
	get_shader_matrix_ids, 120	half_float, 44
	get shader program id, 120	half_float::half, 104
	make mesh, 120	operator>=
	Mesh, 119	half_float, 43
	render, 120	half_float::half, 103
	resize, 121	operator+
	rotate, 121	half_float, 39
	scale, 121	half_float::detail::f31, 67
	set_mesh_type, 121	half_float::half, 99
	set_model_view_matrix, 122	operator++
	set_shader, 122	half_float::half, 72, 73
	translate, 122	operator+=
	udit::Mesh, 124	half_float::half, 73
	update, 122	operator-
mod		half_float, 40
	half.hpp, 191	half_float::detail::f31, 67
mod	• •	half_float::half, 100
	half_float, 36	operator
	half_float::half, 96	half_float::half, 74
mou	se_sensitivity	operator-=
	-	

half_float::half, 74, 75	half_float, 46
operator/	half_float::half, 106
half_float, 41	rotate
half_float::detail::f31, 68	Mesh, 121
half_float::half, 100	udit::Mesh, 127
operator/=	round
half_float::half, 75, 76	half_float, 46
operator=	half_float::half, 106
half_float::half, 76	rounded
operator==	half.hpp, 193
half_float, 42	rsqrt
half_float::half, 102	half_float, 46
operator*	half_float::half, 106
half_float, 39	scalbln
half_float::detail::f31, 67	half_float, 47
half_float::half, 99	half_float::half, 107
operator*=	scalbn
half_float::half, 71, 72	half_float, 47
overflow	scale
half.hpp, 192	Mesh, 121
pitch	udit::Mesh, 127
Camera, 59	Scene, 137
Plane, 131	render, 138
Plane, 133	resize, 138
udit::Plane, 136	Scene, 137
pole	set_lights, 138
half.hpp, 192	set_projection_matrix, 138
position	set_view_matrix, 139
position	
Camera 59	udit::Scene. 140
Camera, 59	udit::Scene, 140 update, 139
pow	update, 139
pow half_float, 44	update, 139 select
pow half_float, 44 half_float::half, 104	update, 139 select half.hpp, 194
pow half_float, 44 half_float::half, 104 process_keyboard	update, 139 select half.hpp, 194 send_to_shader
pow half_float, 44 half_float::half, 104 process_keyboard Camera, 58	update, 139 select half.hpp, 194 send_to_shader udit::Light, 117
pow half_float, 44 half_float::half, 104 process_keyboard Camera, 58 process_mouse_movement	update, 139 select half.hpp, 194 send_to_shader udit::Light, 117 set_lights
pow half_float, 44 half_float::half, 104 process_keyboard Camera, 58	update, 139 select half.hpp, 194 send_to_shader udit::Light, 117 set_lights Scene, 138
pow half_float, 44 half_float::half, 104 process_keyboard Camera, 58 process_mouse_movement	update, 139 select half.hpp, 194 send_to_shader udit::Light, 117 set_lights Scene, 138 udit::Scene, 141
pow half_float, 44 half_float::half, 104 process_keyboard Camera, 58 process_mouse_movement Camera, 58 raise	update, 139 select half.hpp, 194 send_to_shader udit::Light, 117 set_lights Scene, 138 udit::Scene, 141 set_mesh_type
pow half_float, 44 half_float::half, 104 process_keyboard Camera, 58 process_mouse_movement Camera, 58	update, 139 select half.hpp, 194 send_to_shader udit::Light, 117 set_lights Scene, 138 udit::Scene, 141 set_mesh_type Mesh, 121
pow half_float, 44 half_float::half, 104 process_keyboard Camera, 58 process_mouse_movement Camera, 58 raise half.hpp, 193	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
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	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
pow half_float, 44 half_float::half, 104 process_keyboard Camera, 58 process_mouse_movement Camera, 58 raise half.hpp, 193 remainder	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
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	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
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	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
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	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
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	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
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	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
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	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
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	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
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	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
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::half, 105 remder Mesh, 120 Scene, 138 udit::Mesh, 126 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
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::half, 105 remder Mesh, 120 Scene, 138 udit::Mesh, 126 udit::Scene, 140 resize	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, 122 udit::Mesh, 128
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;:half, 105 render Mesh, 120 Scene, 138 udit::Mesh, 126 udit::Scene, 140 resize Mesh, 121	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
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::half, 105 render Mesh, 120 Scene, 138 udit::Mesh, 126 udit::Scene, 140 resize Mesh, 121 Scene, 138 udit::Mesh, 126	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
pow 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::half, 105 render Mesh, 120 Scene, 138 udit::Mesh, 126 udit::Scene, 140 resize Mesh, 121 Scene, 138 udit::Mesh, 126 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
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::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	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
pow 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::half, 105 render Mesh, 120 Scene, 138 udit::Mesh, 126 udit::Scene, 140 resize Mesh, 121 Scene, 138 udit::Mesh, 126 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_type	half_float, 50
Texture, 160	half_float::half, 109
udit::Texture, 162	std, 52
set_view_matrix	std::numeric_limits< half_float::half >, 128
Scene, 139	
udit::Scene, 141	tan
Shader, 142	half_float, 50
\sim Shader, 143	half_float::half, 109
compile_shaders, 144	tangent_post
get_model_view_matrix_id, 144	half.hpp, 196
get_name, 144	tanh
get_normal_matrix_id, 144	half_float, 50
get_program_id, 144	half_float::half, 110
get_projection_matrix_id, 145	TERRAIN
has_textures, 145	Shader.hpp, 246
make_shader, 145	Texture, 158
set_name, 145	bind, 159
set_texture, 147	is_loaded, 159
set_texture_scale, 147	load_texture, 159
Shader, 143	set_type, 160
udit::Shader, 149	Texture, 159
use, 147	udit::Texture, 161
Shader.hpp	unbind, 160
DEFAULT, 246	Texture.hpp
GEOMETRY, 246	COLOR, 250
ShaderType, 246	HEIGHT, 250
SINGLE TEXTURE, 246	Texture_Type, 249
SKYBOX, 246	Texture_Type
TERRAIN, 246	Texture.hpp, 249
ShaderType	tgamma
Shader.hpp, 246	half_float, 51
sign_mask	half_float::half, 110
half.hpp, 194	translate
signal	Mesh, 122
half.hpp, 194, 195	udit::Mesh, 128
signbit	trunc
half float, 48	half_float, 51
half_float::half, 107	half float::half, 111
sin	,
half float, 48	udit::Cube, 62
half_float::half, 108	Cube, 63, 64
	udit::Light, 115
Sincos half han 106	Light, 116
half.hpp, 196	make_light, 116
half_float, 49	send to shader, 117
half_float::half, 108	udit::Mesh, 123
SINGLE_TEXTURE	∼Mesh, 125
Shader.hpp, 246	create mesh, 125
sinh	get_model_view_matrix, 125
half_float, 49	get_shader_matrix_ids, 125
half_float::half, 108	get_shader_program_id, 125
SKYBOX	make mesh, 126
Shader.hpp, 246	Mesh, 124
Skybox, 152	render, 126
getCubemapTexture, 155	resize, 126
Skybox, 154	rotate, 127
udit::Skybox, 157	scale, 127
sqrt	set_mesh_type, 127
half.hpp, 196	set_model_view_matrix, 127
	Set_IIIOuei_view_IIIatiiX, 127

```
set_shader, 128
                                                            Window, 164
    translate, 128
                                                       Window::OpenGL_Context_Settings, 131
     update, 128
                                                       world_up
udit::Plane, 134
                                                            Camera, 60
     Plane, 136
                                                       yaw
udit::Scene, 139
                                                            Camera, 60
     render, 140
     resize, 140
                                                       zoom
     Scene, 140
                                                            Camera, 60
    set_lights, 141
    set_projection_matrix, 141
     set_view_matrix, 141
    update, 141
udit::Shader, 147
     \simShader, 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
```