GL_Scene

Generated by Doxygen 1.13.2

7 File Documentation **165**

Chapter 1

Namespace Index

1.1 Namespace List

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

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

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

Chapter 5

Namespace Documentation

5.1 half_float Namespace Reference

Classes

- class half

Functions

Comparison operators

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

Arithmetic operators

- HALF_CONSTEXPR half operator+ (half arg)
- HALF_CONSTEXPR half operator- (half arg)
- half operator+ (half x, half y)
- half operator- (half x, half y)
- half operator∗ (half x, half y)
- half operator/ (half x, half y)

Input and output

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

Basic mathematical operations

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

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

Exponential functions

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

Power functions

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

Trigonometric functions

- void sincos (half arg, half ∗sin, half ∗cos)
- half sin (half arg)
- half cos (half arg)
- half tan (half arg)
- half asin (half arg)
- half acos (half arg)
- half atan (half arg)
- half atan2 (half y, half x)

Hyperbolic functions

- half sinh (half arg)
- half cosh (half arg)
- half tanh (half arg)
- half asinh (half arg)
- half acosh (half arg)
- half atanh (half arg)

Error and gamma functions

- half erf (half arg)
- half erfc (half arg)
- half lgamma (half arg)
- half tgamma (half arg)

Rounding

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

- half round (half arg)
- long lround (half arg)
- half rint (half arg)
- long lrint (half arg)
- half nearbyint (half arg)

Floating point manipulation

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

Floating point classification

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

Comparison

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

Casting

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

Error handling

- int feclearexcept (int excepts)
- int fetestexcept (int excepts)
- int feraiseexcept (int excepts)
- int fegetexceptflag (int ∗flagp, int excepts)
- int fesetexceptflag (const int ∗flagp, int excepts)
- void fethrowexcept (int excepts, const char ∗msg="")

5.1.1 Detailed Description

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

5.1.2 Function Documentation

5.1.2.1 abs()

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

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

Parameters

arg	operand

Returns

absolute value of *arg*

5.1.2.2 acos()

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

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

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

Parameters

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()

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

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

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

Parameters

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()

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

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

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

Parameters

arg	function argument

Returns

arc tangent value of *arg*

Exceptions

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

5.1.2.7 atan2()

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

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

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

Parameters

y	numerator
x	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()

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

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

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

Parameters

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()

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

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

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

Parameters

arg	function argument

Returns

cubic root of *arg*

Exceptions

FE_INVALID	for signaling NaN
FE_INEXACT	according to rounding

5.1.2.10 ceil()

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

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

Parameters

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()

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

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

Parameters

x	value to change sign for
y	value to take sign from

Returns

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

5.1.2.12 cos()

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

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

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

Parameters

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`.

Parameters

arg	function argument

Returns

hyperbolic cosine value of *arg*

Exceptions

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

5.1.2.14 erf()

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

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

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

Parameters

arg	function argument

Returns

error function value of *arg*

Exceptions

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

5.1.2.15 erfc()

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

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

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

Parameters

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()

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

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

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

Parameters

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

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

5.1.2.19 fabs()

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

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`.

Parameters

x	first operand
y	second operand

Returns

x - *y* or 0 if difference negative

Exceptions

FE↩ _...	according to operator-(half,half)

5.1.2.21 feclearexcept()

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

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

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

Parameters

excepts	OR of exceptions to clear

Return values

0	all selected flags cleared successfully

5.1.2.22 fegetexceptflag()

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

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

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

Parameters

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

Return values

0	for success

5.1.2.23 feraiseexcept()

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

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

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

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

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

Parameters

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

Return values

0	for success

5.1.2.25 fetestexcept()

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

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

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

Parameters

excepts	OR of exceptions to test

Returns

OR of selected exceptions if raised

5.1.2.26 fethrowexcept()

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

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

Parameters

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*

Exceptions

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

x	first operand
y	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()

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

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

Parameters

x	first operand
y	second operand

Returns

maximum of operands, ignoring quiet NaNs

Exceptions

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

5.1.2.30 fmin()

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

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

Parameters

x	first operand
y	second operand

Returns

minimum of operands, ignoring quiet NaNs

Exceptions

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

5.1.2.31 fmod()

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

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

Parameters

x	first operand
y	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

5.1.2.32 fpclassify()

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

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

5.1.2.33 frexp()

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

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

Parameters

arg	number to decompress
exp	address to store exponent at

Returns

significant in range [0.5, 1)

Exceptions

FE_INVALID	for signaling NaN

5.1.2.34 half_cast() [1/2]

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

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

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

arg	value to cast

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]

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

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

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

Exceptions

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

5.1.2.36 hypot() [1/2]

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

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

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

Parameters

x	first argument
y	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

x	first argument
y	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,...UNDERFLOW,...INEXACT	according to rounding of the final square root

5.1.2.38 ilogb()

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

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

Parameters

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

5.1.2.39 isfinite()

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

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

Parameters

arg	number to check

Return values

true	if neither infinity nor NaN
false	else

5.1.2.40 isgreater()

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

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

Parameters

x	first operand
y	second operand

Return values

true	if *x* greater than *y*
false	else

5.1.2.41 isgreaterequal()

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

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

Parameters

x	first operand
y	second operand

Return values

true	if *x* greater equal *y*
false	else

5.1.2.42 isinf()

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

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

Parameters

arg	number to check

Return values

true	for positive or negative infinity
false	else

5.1.2.43 isless()

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

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

Parameters

x	first operand
y	second operand

Return values

true	if *x* less than *y*
false	else

5.1.2.44 islessequal()

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

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

Parameters

x	first operand
y	second operand

Return values

true	if *x* less equal *y*
false	else

5.1.2.45 islessgreater()

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

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

Parameters

x	first operand
y	second operand

Return values

true	if either less or greater
false	else

5.1.2.46 isnan()

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

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

Parameters

arg	number to check

Return values

true	for NaNs
false	else

5.1.2.47 isnormal()

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

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

Parameters

arg	number to check

Return values

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

5.1.2.48 isunordered()

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

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

Parameters

x	first operand
y	second operand

Return values

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

5.1.2.49 ldexp()

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

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

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

Parameters

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()

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

Natural logarithm of gamma function. This function may be 1 ULP off the correctly rounded exact result for any rounding mode in ∼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

5.1.2.51 log()

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

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

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

Parameters

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()

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

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

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

Parameters

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 ∼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

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()

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

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

5.1.2.56 lrint()

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

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

Parameters

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 lround()

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

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()

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

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

Parameters

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()

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

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()

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

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

5.1.2.62 nexttoward()

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

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

Parameters

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

5.1.2.63 operator"!=()

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

Comparison for inequality.

Parameters

x	first operand
y	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∗()

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

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

Parameters

x	left operand
y	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]

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

Identity.

Parameters

arg	operand

Returns

unchanged operand

5.1.2.66 operator+() [2/2]

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

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

Parameters

x	left operand
y	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]

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

Negation.

Parameters

arg	operand

Returns

negated operand

5.1.2.68 operator-() [2/2]

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

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

Parameters

x	left operand
y	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

5.1.2.69 operator/()

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

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

Parameters

x	left operand
y	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<()

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

Comparison for less than.

Parameters

x	first operand
y	second operand

Return values

true	if *x* less than *y*
false	else

Exceptions

FE_INVALID	if *x* or *y* is NaN

5.1.2.71 operator<<()

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

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

Parameters

out	output stream to write into
arg	half expression to write

Returns

reference to output stream

5.1.2.72 operator<=()

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

Comparison for less equal.

Parameters

x	first operand
y	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==()

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

Comparison for equality.

Parameters

x	first operand
y	second operand

Return values

true	if operands equal
false	else

Exceptions

FE_INVALID	if *x* or *y* is NaN

5.1.2.74 operator>()

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

Comparison for greater than.

Parameters

x	first operand
y	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>=()

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

Comparison for greater equal.

Parameters

x	first operand
y	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>>()

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

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

Parameters

in	input stream to read from
arg	half to read into

Returns

reference to input stream

Exceptions

FE_OVERFLOW,...UNDERFLOW,...INEXACT	according to rounding

5.1.2.77 pow()

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

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

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

Parameters

x	base
y	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

5.1.2.78 remainder()

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

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

Parameters

x	first operand
y	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()

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

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

Parameters

x	first operand
y	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

5.1.2.80 rint()

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

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

Parameters

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()

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

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()

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

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

Parameters

arg	function argument

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()

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

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

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

Parameters

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

5.1.2.84 scalbn()

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

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

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

Parameters

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()

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

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()

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

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

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

Parameters

arg	function argument

Returns

sine value of *arg*

Exceptions

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()

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

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

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

Parameters

arg	function argument

Returns

hyperbolic sine value of *arg*

Exceptions

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

5.1.2.89 sqrt()

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

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

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

Parameters

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`.

Parameters

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()

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

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

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

Parameters

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()

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

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

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:

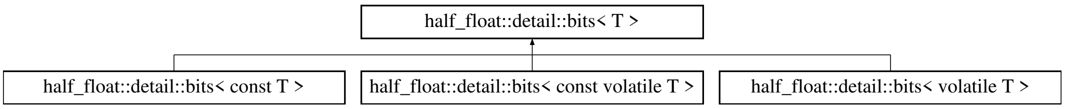
- GL_Scene/half.hpp

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 >:


half_float::detail::bits< T >
half_float::detail::bits< const T >
half_float::detail::bits< const volatile T >
half_float::detail::bits< volatile T >

Public Types

- typedef unsigned char **type**

6.2.1 Detailed Description

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

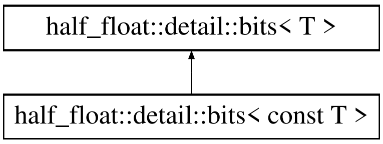
Type traits for floating-point bits.

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

- GL_Scene/half.hpp

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

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


half_float::detail::bits< T >
half_float::detail::bits< const T >

Public Types

- typedef unsigned char **type**

Public Types inherited from half_float::detail::bits< T >

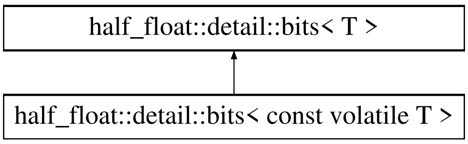
- typedef unsigned char **type**

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 >:


half_float::detail::bits< T >
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:

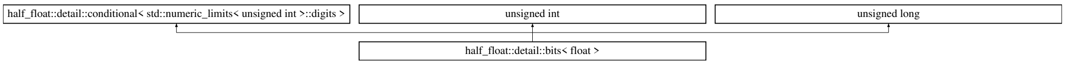
- GL_Scene/half.hpp

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 >:


half_float::detail::conditional< std::numeric_limits< unsigned int >::digits >
unsigned int
unsigned long
half_float::detail::bits< float >

Public Types

- typedef unsigned char **type**

Public Types inherited from half_float::detail::conditional< std::numeric_limits< unsigned int >::digits >

- typedef T **type**

6.6.1 Detailed Description

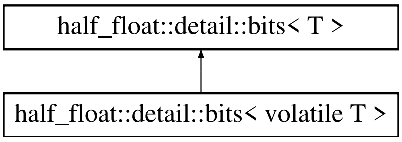
Unsigned integer of (at least) 32 bits width.

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

- GL_Scene/half.hpp

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

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


half_float::detail::bits< T >
half_float::detail::bits< volatile T >

Public Types

- typedef unsigned char **type**

Public Types inherited from half_float::detail::bits< T >

- typedef unsigned char **type**

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

- GL_Scene/half.hpp

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

Helper for tag dispatching.

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

6.8.1 Detailed Description

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

Helper for tag dispatching.

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

- GL_Scene/half.hpp

6.9 Camera Class Reference

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

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

Public Member Functions

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

  Constructor de la cámara.
- glm::mat4 get_view_matrix () const

  Obtiene la matriz de vista de la cámara.
- void process_keyboard (CameraMovement direction, float delta_time)

  Procesa la entrada de teclado para mover la cámara.
- void process_mouse_movement (float x_offset, float y_offset, bool constraint_pitch=true)

  Procesa el movimiento del ratón para rotar la cámara.

Public Attributes

- glm::vec3 position

  Posición actual de la cámara.
- glm::vec3 front

  Dirección hacia la cual está mirando la cámara.
- glm::vec3 up

  Vectores de orientación de la cámara en el eje Y (arriba).
- glm::vec3 right

  Vectores de la orientación de la cámara en el eje X (derecha).
- glm::vec3 world_up

  Dirección "arriba" global.
- float yaw

  Ángulo de orientación de la cámara alrededor del eje Y.
- float pitch

  Ángulo de orientación de la cámara alrededor del eje X.
- float movement_speed

  Velocidad de movimiento de la cámara.
- float mouse_sensitivity

  Sensibilidad al movimiento del ratón.
- float zoom

  Nivel de zoom de la cámara.

6.9.1 Detailed Description

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

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

6.9.2 Constructor & Destructor Documentation

6.9.2.1 Camera()

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

Constructor de la cámara.

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

Parameters

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()

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

Procesa la entrada de teclado para mover la cámara.

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

Parameters

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()

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

Procesa el movimiento del ratón para rotar la cámara.

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

Parameters

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.

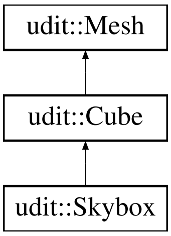
La clase Cube crea y gestiona un cubo 3D. Ofrece constructores para crear un cubo con un tamaño específico y con la opción de invertir las normales. Hereda de la clase Mesh y aprovecha sus funcionalidades para el procesamiento y renderizado del cubo en un entorno OpenGL.

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

inverted	Si es `true`, las normales del cubo se invierten.

6.13.1.3 Cube() [3/4]

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

Constructor con tamaño especificado.

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

Parameters

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()

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

Constructor que inicializa el EventHandler con una referencia a la cámara.

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

Parameters

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]

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

Constructor.

Parameters

mant	mantissa as 1.31
e	exponent

6.15.2.2 f31() [2/2]

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

Constructor.

Parameters

abs	unsigned half-precision value

6.15.3 Friends And Related Symbol Documentation

6.15.3.1 operator*

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

Multiplication operator.

Parameters

a	first operand
b	second operand

Returns

$a * b$

6.15.3.2 operator+

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

Addition operator.

Parameters

a	first operand
b	second operand

Returns

$a + b$

6.15.3.3 operator-

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

Subtraction operator.

Parameters

a	first operand
b	second operand

Returns

a - *b*

6.15.3.4 operator/

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

Division operator.

Parameters

a	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

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

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

6.16.1 Detailed Description

Half-precision floating-point type. This class implements an IEEE-conformant half-precision floating-point type with the usual arithmetic operators and conversions. It is implicitly convertible to single-precision floating-point, which makes artihmetic 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]

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

Conversion constructor.

Parameters

rhs	float to convert

Exceptions

FE_OVERFLOW,...UNDERFLOW,...INEXACT	according to rounding

6.16.3 Member Function Documentation

6.16.3.1 operator float()

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

Conversion to single-precision.

Returns

single precision value representing expression value

6.16.3.2 operator∗=() [1/2]

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

Arithmetic assignment.

Parameters

rhs	single-precision value to multiply with

Returns

reference to this half

Exceptions

FE↩ _...	according to operator=()

6.16.3.3 operator∗=() [2/2]

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

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]

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

Postfix increment.

Returns

non-incremented half value

Exceptions

FE← _...	according to operator+(half,half)

6.16.3.6 operator+=() [1/2]

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

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]

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

Arithmetic assignment.

Template Parameters

T	type of concrete half expression

Parameters

rhs	half expression 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]

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

Postfix decrement.

Returns

non-decremented half value

Exceptions

FE← _...	according to operator-(half,half)

6.16.3.10 operator-=() [1/2]

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

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]

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

Arithmetic assignment.

Template Parameters

T	type of concrete half expression

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]

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

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]

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

Arithmetic assignment.

Template Parameters

T	type of concrete half expression

Parameters

rhs	half expression to divide by

Returns

reference to this half

Exceptions

FE↩ _...	according to operator/(half,half)

6.16.3.14 operator=()

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

Assignment operator.

Parameters

rhs	single-precision value to copy from

Returns

reference to this half

Exceptions

FE_OVERFLOW,...UNDERFLOW,...INEXACT	according to rounding

6.16.4 Friends And Related Symbol Documentation

6.16.4.1 acos

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

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

See also: Documentation for std::acos.

Parameters

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

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

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

Parameters

y	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

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

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

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

Parameters

x	value to change sign for
y	value to take sign from

Returns

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

6.16.4.11 cos

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

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

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

Parameters

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

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::expm1`.

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

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

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

Parameters

x	first operand
y	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

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

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

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

Parameters

x	first operand
y	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

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

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

Parameters

x	first operand
y	second operand

Returns

maximum of operands, ignoring quiet NaNs

Exceptions

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

6.16.4.23 fmin

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

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

Parameters

x	first operand
y	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 (
            half x,
            half y)   [friend]
```

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

Parameters

x	first operand
y	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

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

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

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

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

Parameters

x	first argument
y	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 (
            half x,
            half y,
            half z)   [friend]
```

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

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

Parameters

x	first argument
y	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,...UNDERFLOW,...INEXACT	according to rounding of the final square root

6.16.4.29 ilogb

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

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

Parameters

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

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

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

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

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

Parameters

x	first operand
y	second operand

Return values

true	if *x* greater than *y*
false	else

6.16.4.32 isgreaterequal

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

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

Parameters

x	first operand
y	second operand

Return values

true	if *x* greater equal *y*
false	else

6.16.4.33 isinf

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

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

Parameters

arg	number to check

Return values

true	for positive or negative infinity
false	else

6.16.4.34 isless

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

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

Parameters

x	first operand
y	second operand

Return values

true	if *x* less than *y*
false	else

6.16.4.35 islessequal

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

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

Parameters

x	first operand
y	second operand

Return values

true	if *x* less equal *y*
false	else

6.16.4.36 islessgreater

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

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

Parameters

x	first operand
y	second operand

Return values

true	if either less or greater
false	else

6.16.4.37 isnan

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

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

Parameters

arg	number to check

Return values

true	for NaNs
false	else

6.16.4.38 isnormal

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

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

6.16.4.39 lgamma

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

Natural logarithm of gamma function. This function may be 1 ULP off the correctly rounded exact result for any rounding mode in ∼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

```
half log10 (
          half arg)  [friend]
```

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 ∼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

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

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 lrint

```
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	if value had to be rounded

6.16.4.46 lround

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

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

Parameters

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

```
half modf (
             half arg,
             half * iptr)   [friend]
```

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

Parameters

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

```
half nanh (
            const char * arg)  [friend]
```

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

```
half nextafter (
            half from,
            half to)  [friend]
```

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"!=

```
HALF_CONSTEXPR_NOERR bool operator!= (
            half x,
            half y)  [friend]
```

Comparison for inequality.

Parameters

x	first operand
y	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*

```
half operator* (
            half x,
            half y)  [friend]
```

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

Parameters

x	left operand
y	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+ (
            half x,
            half y)  [friend]
```

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

Parameters

x	left operand
y	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]

```
HALF_CONSTEXPR half operator- (
            half arg)  [friend]
```

Negation.

Parameters

arg	operand

Returns

negated operand

6.16.4.56 operator- [2/2]

```
half operator- (
            half x,
            half y)  [friend]
```

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

Parameters

x	left operand
y	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/

```
half operator/ (
            half x,
            half y)  [friend]
```

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

Parameters

x	left operand
y	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<

```
HALF_CONSTEXPR_NOERR bool operator< (
            half x,
            half y)  [friend]
```

Comparison for less than.

Parameters

x	first operand
y	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<<

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

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

Parameters

out	output stream to write into
arg	half expression to write

Returns

reference to output stream

6.16.4.60 operator<=

```
HALF_CONSTEXPR_NOERR bool operator<= (
            half x,
            half y)  [friend]
```

Comparison for less equal.

Parameters

x	first operand
y	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==

```
HALF_CONSTEXPR_NOERR bool operator== (
            half x,
            half y)  [friend]
```

Comparison for equality.

Parameters

x	first operand
y	second operand

Return values

true	if operands equal
false	else

Exceptions

FE_INVALID	if *x* or *y* is NaN

6.16.4.62 operator>

```
HALF_CONSTEXPR_NOERR bool operator> (
            half x,
            half y)  [friend]
```

Comparison for greater than.

Parameters

x	first operand
y	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>=

```
HALF_CONSTEXPR_NOERR bool operator>= (
            half x,
            half y)  [friend]
```

Comparison for greater equal.

Parameters

x	first operand
y	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>>

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

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

Parameters

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

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

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

Parameters

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

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

Parameters

x	first operand
y	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

```
half remquo (
            half x,
            half y,
            int * quo)  [friend]
```

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

Parameters

x	first operand
y	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

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

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.

Parameters

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
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

6.16.4.72 signbit

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

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`.

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

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_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:

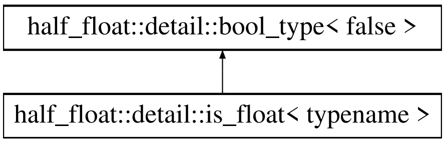
- GL_Scene/half.hpp

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 >:


half_float::detail::bool_type< false >
half_float::detail::is_float< typename >

6.21.1 Detailed Description

template<typename>
struct half_float::detail::is_float< typename >

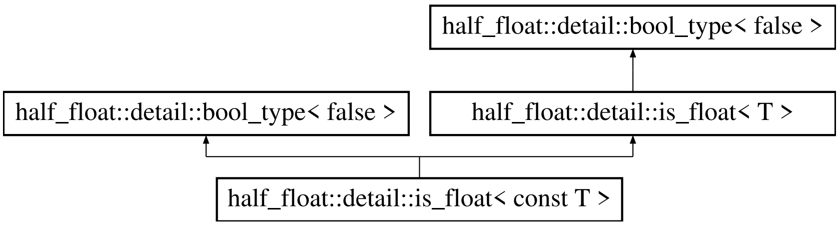
Type traits for floating-point types.

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 >:

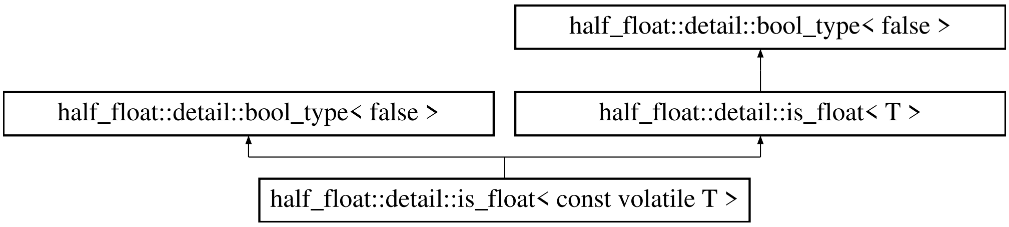

half_float::detail::bool_type< false >
half_float::detail::bool_type< false >
half_float::detail::is_float< T >
half_float::detail::is_float< const T >

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 >:

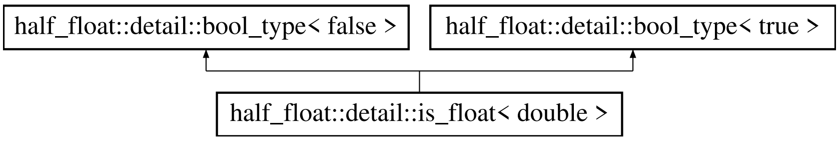

half_float::detail::bool_type< false >
half_float::detail::bool_type< false >
half_float::detail::is_float< T >
half_float::detail::is_float< const volatile T >

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 >:

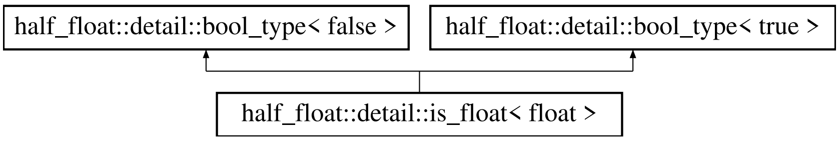

half_float::detail::bool_type< false >
half_float::detail::bool_type< true >
half_float::detail::is_float< double >

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 >:

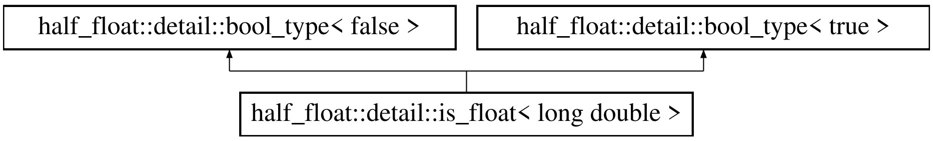

half_float::detail::bool_type< false >
half_float::detail::bool_type< true >
half_float::detail::is_float< float >

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 >:

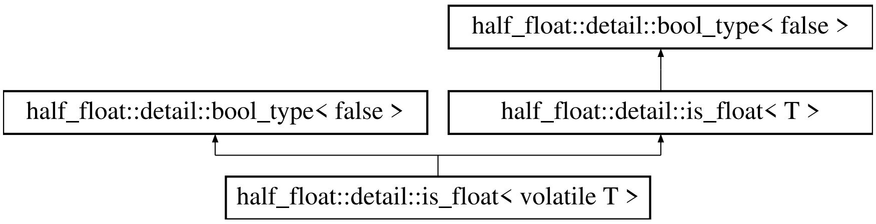

half_float::detail::bool_type< false >
half_float::detail::bool_type< true >
half_float::detail::is_float< long double >

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 >:


half_float::detail::bool_type< false >
half_float::detail::bool_type< false >
half_float::detail::is_float< T >
half_float::detail::is_float< volatile T >

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

- GL_Scene/half.hpp

6.28 udit::Light Class Reference

Clase que representa una fuente de luz en la escena.

```
#include <Light.hpp>
```

Public Member Functions

- Light (const glm::vec3 &pos, const glm::vec3 &col, float ambient, float diffuse)

  Constructor de la clase Light.
- void send_to_shader (GLuint program_id) const

  Envía los parámetros de la luz al shader.

Static Public Member Functions

- static std::shared_ptr< Light > make_light (const glm::vec3 &pos, const glm::vec3 &col, float ambient, float diffuse)

  Crea una luz a partir de los parámetros especificados.

6.28.1 Detailed Description

Clase que representa una fuente de luz en la escena.

La clase `Light` es responsable de definir las características básicas de una fuente de luz, tales como su posición, color y las intensidades de la luz ambiental y difusa. Esta clase se utiliza para enviar la información de la luz a los shaders en OpenGL para que los efectos de luz sean aplicados en la escena 3D.

6.28.2 Constructor & Destructor Documentation

6.28.2.1 Light()

```
Light::Light (
            const glm::vec3 & pos,
            const glm::vec3 & col,
            float ambient,
            float diffuse)
```

Constructor de la clase Light.

Este constructor inicializa los parámetros de la luz con valores específicos para su posición, color y las intensidades de luz ambiental y difusa.

Parameters

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()

```
std::shared_ptr< Light > Light::make_light (
            const glm::vec3 & pos,
            const glm::vec3 & col,
            float ambient,
            float diffuse) [static]
```

Crea una luz a partir de los parámetros especificados.

Esta función estática facilita la creación de un objeto `Light` compartido (`shared_ptr`) con los valores de posición, color e intensidades de luz ambiental y difusa.

Parameters

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()

```
void Light::send_to_shader (
             GLuint program_id) const
```

Envía los parámetros de la luz al shader.

Esta función toma los parámetros de la luz (posición, color, intensidad) y los envía al shader especificado a través de su programa de OpenGL. Esto permite que la luz sea utilizada en los cálculos de sombreado dentro del pipeline de gráficos.

Parameters

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

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

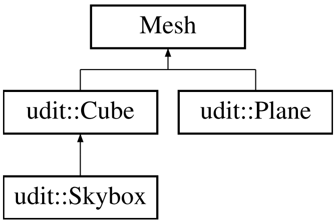
- GL_Scene/Light.hpp
- GL_Scene/Light.cpp

6.29 Mesh Class Reference

Clase que representa una malla 3D.

`#include <Mesh.hpp>`

Inheritance diagram for Mesh:



Public Member Functions

- **Mesh** ()

  Constructor por defecto.
- Mesh (std::string &path)

  Constructor que carga una malla desde un archivo.
- virtual ∼Mesh ()

  Destructor de la clase.
- virtual void translate (glm::vec3 translation)

  Realiza una traslación de la malla.
- virtual void rotate (glm::vec3 rotation, float angle)

  Rota la malla.
- virtual void scale (glm::vec3 scale)

  Escala la malla.
- virtual void update ()

  Actualiza la malla.
- virtual void render (glm::mat4 view_matrix)

  Renderiza la malla.
- virtual void resize (glm::mat4 projection_matrix)

  Ajusta la matriz de proyección.
- virtual void set_shader (std::shared_ptr< udit::Shader > shader)

  Asocia un shader a la malla.
- GLuint get_shader_program_id () const

  Obtiene el ID del programa del shader asociado.
- std::vector< GLint > get_shader_matrix_ids ()

  Obtiene los IDs de las matrices del shader asociadas a la malla.
- glm::mat4 get_model_view_matrix () const

  Obtiene la matriz de transformación del modelo.
- void set_model_view_matrix (glm::mat4 matrix)

  Establece la matriz de transformación del modelo.
- void set_mesh_type (MeshType type)

  Establece el tipo de malla.

Static Public Member Functions

- static std::shared_ptr< Mesh > make_mesh (MeshType type, const std::string &path="")

  Crea una malla de un tipo específico.

Protected Member Functions

- void create_mesh (std::string mesh_name="")

  Crea los VBOs y el VAO necesarios para la malla.

Protected Attributes

- std::vector< glm::vec3 > **coordinates**

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

  Número total de vértices de la malla.

6.29.1 Detailed Description

Clase que representa una malla 3D.

La clase `Mesh` es la base para representar mallas 3D en OpenGL. Contiene todos los atributos y funciones necesarias para cargar, gestionar y renderizar mallas con vértices, normales, colores, coordenadas de textura y los índices que definen la topología de la malla. Esta clase también incluye funciones para transformar la malla (traslación, rotación, escala) y para actualizar y renderizar la malla en la escena.

6.29.2 Constructor & Destructor Documentation

6.29.2.1 Mesh()

```
udit::Mesh::Mesh (
            std::string & path)
```

Constructor que carga una malla desde un archivo.

Este constructor carga los datos de la malla (coordenadas, normales, colores, etc.) desde un archivo y los almacena en los atributos correspondientes.

Parameters

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()

```
void udit::Mesh::create_mesh (
            std::string mesh_name = "")  [protected]
```

Crea los VBOs y el VAO necesarios para la malla.

Parameters

mesh_name	Nombre de la malla a crear.

6.29.3.2 get_model_view_matrix()

```
glm::mat4 udit::Mesh::get_model_view_matrix () const  [inline]
```

Obtiene la matriz de transformación del modelo.

Returns

La matriz de transformación del modelo.

6.29.3.3 get_shader_matrix_ids()

```
std::vector< GLint > udit::Mesh::get_shader_matrix_ids ()
```

Obtiene los IDs de las matrices del shader asociadas a la malla.

Devuelve los IDs de las matrices necesarias para renderizar la malla en el shader.

Returns

Un vector con los IDs de las matrices.

6.29.3.4 get_shader_program_id()

```
GLuint udit::Mesh::get_shader_program_id () const
```

Obtiene el ID del programa del shader asociado.

Returns

El ID del programa de shader asociado a la malla.

6.29.3.5 make_mesh()

```
std::shared_ptr< Mesh > udit::Mesh::make_mesh (
            MeshType type,
            const std::string & path = "")  [static]
```

Crea una malla de un tipo específico.

Este método estático permite crear una malla de un tipo específico, como terreno, malla básica, o malla cargada desde un archivo.

Parameters

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()

```
void udit::Mesh::render (
            glm::mat4 view_matrix)  [virtual]
```

Renderiza la malla.

Función de renderizado de la malla en el bucle principal.

Utiliza el shader asociado y la matriz de vista para renderizar la malla.

Parameters

view_matrix	Matriz de vista.

6.29.3.7 resize()

```
void udit::Mesh::resize (
            glm::mat4 projection_matrix)  [virtual]
```

Ajusta la matriz de proyección.

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

Parameters

projection_matrix	Matriz de proyección.

6.29.3.8 rotate()

```
void udit::Mesh::rotate (
            glm::vec3 rotation,
            float angle)  [virtual]
```

Rota la malla.

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

Parameters

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

6.29.3.9 scale()

```
void udit::Mesh::scale (
            glm::vec3 scale)  [virtual]
```

Escala la malla.

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

Parameters

scale	Factor de escala.

6.29.3.10 set_mesh_type()

```
void udit::Mesh::set_mesh_type (
            MeshType type)  [inline]
```

Establece el tipo de malla.

Parameters

type	Tipo de malla.

6.29.3.11 set_model_view_matrix()

```
void udit::Mesh::set_model_view_matrix (
            glm::mat4 matrix)  [inline]
```

Establece la matriz de transformación del modelo.

Parameters

matrix	Nueva matriz de transformación del modelo.

6.29.3.12 set_shader()

```
void udit::Mesh::set_shader (
            std::shared_ptr< udit::Shader > shader)  [virtual]
```

Asocia un shader a la malla.

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

Parameters

shader	Puntero al shader a asociar.

6.29.3.13 translate()

```
void udit::Mesh::translate (
            glm::vec3 translation)  [virtual]
```

Realiza una traslación de la malla.

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

Parameters

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:

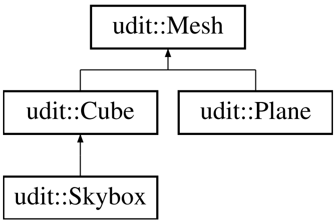
- GL_Scene/Mesh.hpp
- GL_Scene/Mesh.cpp

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()

```
udit::Mesh::Mesh (
             std::string & path)
```

Constructor que carga una malla desde un archivo.

Este constructor carga los datos de la malla (coordenadas, normales, colores, etc.) desde un archivo y los almacena en los atributos correspondientes.

Parameters

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()

```
void udit::Mesh::create_mesh (
             std::string mesh_name = "")  [protected]
```

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()

```
std::shared_ptr< Mesh > udit::Mesh::make_mesh (
            MeshType type,
            const std::string & path = "") [static]
```

Crea una malla de un tipo específico.

Este método estático permite crear una malla de un tipo específico, como terreno, malla básica, o malla cargada desde un archivo.

Parameters

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()

```
void udit::Mesh::render (
            glm::mat4 view_matrix) [virtual]
```

Renderiza la malla.

Función de renderizado de la malla en el bucle principal.

Utiliza el shader asociado y la matriz de vista para renderizar la malla.

Parameters

view_matrix	Matriz de vista.

6.30.3.7 resize()

```
void udit::Mesh::resize (
            glm::mat4 projection_matrix) [virtual]
```

Ajusta la matriz de proyección.

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

Parameters

projection_matrix	Matriz de proyección.

6.30.3.8 rotate()

```
void udit::Mesh::rotate (
            glm::vec3 rotation,
            float angle)  [virtual]
```

Rota la malla.

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

Parameters

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

6.30.3.9 scale()

```
void udit::Mesh::scale (
            glm::vec3 scale)  [virtual]
```

Escala la malla.

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

Parameters

scale	Factor de escala.

6.30.3.10 set_mesh_type()

```
void udit::Mesh::set_mesh_type (
            MeshType type)  [inline]
```

Establece el tipo de malla.

Parameters

type	Tipo de malla.

6.30.3.11 set_model_view_matrix()

```
void udit::Mesh::set_model_view_matrix (
            glm::mat4 matrix)  [inline]
```

Establece la matriz de transformación del modelo.

Parameters

matrix	Nueva matriz de transformación del modelo.

6.30.3.12 set_shader()

```
void udit::Mesh::set_shader (
            std::shared_ptr< udit::Shader > shader)  [virtual]
```

Asocia un shader a la malla.

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

Parameters

shader	Puntero al shader a asociar.

6.30.3.13 translate()

```
void udit::Mesh::translate (
            glm::vec3 translation)  [virtual]
```

Realiza una traslación de la malla.

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

Parameters

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:

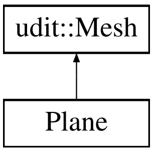
- GL_Scene/Window.hpp

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.

Protected Attributes inherited from udit::Mesh

- std::vector< glm::vec3 > **coordinates**

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

  Número total de vértices de la malla.

6.34.1 Detailed Description

Clase que representa un plano 3D.

La clase `Plane` hereda de `Mesh` y está diseñada para representar un plano 3D en OpenGL. El plano se define por su ancho, altura, y la cantidad de columnas y filas que tiene. Esta clase permite crear un plano con diferentes configuraciones, ya sea con un tamaño específico o con una distribución de vértices más compleja. El plano es útil para representar superficies planas, como terrenos o fondos.

6.34.2 Constructor & Destructor Documentation

6.34.2.1 Plane() [1/3]

```
udit::Plane::Plane ()
```

Constructor por defecto.

Crea un plano con dimensiones predeterminadas.

6.34.2.2 Plane() [2/3]

```
udit::Plane::Plane (
                float size)
```

Constructor que define el tamaño del plano.

Crea un plano cuadrado con el tamaño especificado.

Parameters

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 horizontal).
rows	Número de filas del plano (resolución vertical).

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

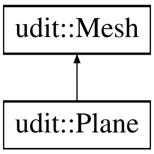
- GL_Scene/Plane.hpp
- GL_Scene/Plane.cpp

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]

```
udit::Plane::Plane (
            float size)
```

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]

```
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 horizontal).
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

Representa una escena 3D con un skybox, terreno, luz y otros elementos.

```
#include <Scene.hpp>
```

Public Member Functions

- Scene (unsigned width, unsigned height)

  Constructor de la escena.
- void update ()

  Actualiza la escena.
- void render ()

  Renderiza la escena.
- void resize (unsigned width, unsigned height)

  Redimensiona la escena.
- void set_view_matrix (const glm::mat4 &view)

  Establece la matriz de vista para la cámara.
- void set_projection_matrix (const glm::mat4 &projection)

  Establece la matriz de proyección para la cámara.
- void set_lights (GLuint shader_program_id)

  Establece las luces en el shader.

6.36.1 Detailed Description

Representa una escena 3D con un skybox, terreno, luz y otros elementos.

La clase `Scene` es responsable de gestionar la representación de una escena 3D, incluyendo los objetos gráficos principales y la iluminación. Los métodos permiten actualizar la escena, renderizarla y ajustar su tamaño.

6.36.2 Constructor & Destructor Documentation

6.36.2.1 Scene()

```
udit::Scene::Scene (
            unsigned width,
            unsigned height)
```

Constructor de la escena.

Constructor.

Inicializa una nueva escena con el ancho y alto especificados.

Parameters

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

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

6.36.3.3 set_lights()

```
void udit::Scene::set_lights (
            GLuint shader_program_id)
```

Establece las luces en el shader.

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

Parameters

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

6.36.3.4 set_projection_matrix()

```
void udit::Scene::set_projection_matrix (
            const glm::mat4 & projection)
```

Establece la matriz de proyección para la cámara.

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

Parameters

projection	Matriz de proyección.

6.36.3.5 set_view_matrix()

```
void udit::Scene::set_view_matrix (
            const glm::mat4 & view)
```

Establece la matriz de vista para la cámara.

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

Parameters

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()

```
void udit::Scene::set_lights (
            GLuint shader_program_id)
```

Establece las luces en el shader.

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

Parameters

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

6.37.3.4 set_projection_matrix()

```
void udit::Scene::set_projection_matrix (
            const glm::mat4 & projection)
```

Establece la matriz de proyección para la cámara.

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

Parameters

projection	Matriz de proyección.

6.37.3.5 set_view_matrix()

```
void udit::Scene::set_view_matrix (
            const glm::mat4 & view)
```

Establece la matriz de vista para la cámara.

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

Parameters

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::vector< std::string > &texture_paths={""}, const std::string &name="")

  Crea un shader.

6.38.1 Detailed Description

Representa un shader program en OpenGL.

La clase `Shader` gestiona la creación y uso de programas de sombreado en OpenGL. Permite compilar los shaders, vincularlos en un programa y usarlos para renderizar objetos en la escena. También proporciona funciones para gestionar texturas y matrices de transformación, como la matriz de modelo-vista, proyección y normales.

6.38.2 Constructor & Destructor Documentation

6.38.2.1 Shader() [1/2]

```
udit::Shader::Shader ()
```

Constructor por defecto.

Crea un objeto `Shader` sin especificar un tipo o fuentes de shader. Este constructor generalmente se usa para crear shaders más tarde con la función `make_shader`.

6.38.2.2 Shader() [2/2]

```
udit::Shader::Shader (
            ShaderType type,
            const std::string & vertex_source,
            const std::string & fragment_source,
            const std::string & name)
```

Constructor para crear un shader con tipos y fuentes especificadas.

Parameters

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::∼Shader ()
```

Destructor.

Libera los recursos asociados al shader.

6.38.3 Member Function Documentation

6.38.3.1 compile_shaders()

```
GLuint udit::Shader::compile_shaders (
            const char * vertex_shader_code,
            const char * fragment_shader_code)
```

Compila los shaders.

Compilador de los shaders construidos.

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

Parameters

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()

```
void udit::Shader::set_name (
             const std::string & name)  [inline]
```

Establece el nombre del shader.

Parameters

name	Nombre del shader.

6.38.3.10 set_texture()

```
void udit::Shader::set_texture (
             const std::shared_ptr< Texture > & texture)
```

Establece una textura para el shader.

Parameters

texture	Puntero a la textura que será asignada al shader.

6.38.3.11 set_texture_scale()

```
void udit::Shader::set_texture_scale (
             float scale)
```

Establece la escala de las texturas asociadas al shader.

Parameters

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.
- Shader (ShaderType type, const std::string &vertex_source, const std::string &fragment_source, const std↩ ::string &name)

  Constructor para crear un shader con tipos y fuentes especificadas.
- ∼Shader ()

  Destructor.
- GLuint compile_shaders (const char ∗vertex_shader_code, const char ∗fragment_shader_code)

  Compila los shaders.
- GLint get_model_view_matrix_id ()

  Obtiene el identificador de la matriz de modelo-vista.
- GLint get_projection_matrix_id ()

  Obtiene el identificador de la matriz de proyección.
- GLint get_normal_matrix_id ()

  Obtiene el identificador de la matriz de normales.
- GLuint get_program_id () const

  Obtiene el identificador del programa de shader.
- void set_texture (const std::shared_ptr< Texture > &texture)

  Establece una textura para el shader.
- void use () const

  Activa y usa el programa de shader.
- void set_texture_scale (float scale)

  Establece la escala de las texturas asociadas al shader.
- bool has_textures ()

  Verifica si el shader tiene texturas asociadas.
- void set_name (const std::string &name)

  Establece el nombre del shader.
- std::string get_name ()

  Obtiene el nombre del shader.

Static Public Member Functions

- static std::shared_ptr< Shader > make_shader (udit::ShaderType type=udit::ShaderType::DEFAULT, const std::string &vertex_shader="", const std::string &fragment_shader="", const std::vector< std::string > &texture_paths={""}, const std::string &name="")

  Crea un shader.

6.39.1 Detailed Description

Representa un shader program en OpenGL.

La clase `Shader` gestiona la creación y uso de programas de sombreado en OpenGL. Permite compilar los shaders, vincularlos en un programa y usarlos para renderizar objetos en la escena. También proporciona funciones para gestionar texturas y matrices de transformación, como la matriz de modelo-vista, proyección y normales.

6.39.2 Constructor & Destructor Documentation

6.39.2.1 Shader() [1/2]

```
udit::Shader::Shader ()
```

Constructor por defecto.

Crea un objeto Shader sin especificar un tipo o fuentes de shader. Este constructor generalmente se usa para crear shaders más tarde con la función make_shader.

6.39.2.2 Shader() [2/2]

```
udit::Shader::Shader (
            ShaderType type,
            const std::string & vertex_source,
            const std::string & fragment_source,
            const std::string & name)
```

Constructor para crear un shader con tipos y fuentes especificadas.

Parameters

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()

```
GLuint udit::Shader::compile_shaders (
            const char * vertex_shader_code,
            const char * fragment_shader_code)
```

Compila los shaders.

Compilador de los shaders construidos.

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

Parameters

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()

```
void udit::Shader::set_name (
             const std::string & name)  [inline]
```

Establece el nombre del shader.

Parameters

name	Nombre del shader.

6.39.3.10 set_texture()

```
void udit::Shader::set_texture (
             const std::shared_ptr< Texture > & texture)
```

Establece una textura para el shader.

Parameters

texture	Puntero a la textura que será asignada al shader.

6.39.3.11 set_texture_scale()

```
void udit::Shader::set_texture_scale (
             float scale)
```

Establece la escala de las texturas asociadas al shader.

Parameters

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:

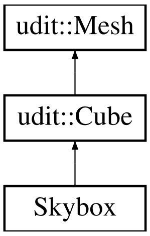
- GL_Scene/Shader.hpp
- GL_Scene/Shader.cpp

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]

```
udit::Skybox::Skybox (
            float size,
            const std::vector< std::string > & faces)
```

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:

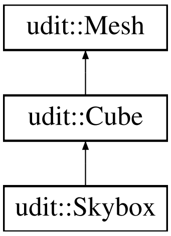
- GL_Scene/Skybox.hpp
- GL_Scene/Skybox.cpp

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]

```
udit::Skybox::Skybox (
          float size,
          const std::vector< std::string > & faces)
```

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.
- ∼**Texture** ()

  Destructor que libera la textura cargada.
- void bind () const

  Enlaza la textura a la unidad de textura actual.
- void unbind () const

  Desenlaza la textura de la unidad de textura.
- void load_texture ()

  Carga la textura desde el archivo especificado.
- void set_type (Texture_Type type)

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

  Indica si la textura ha sido cargada exitosamente.

Public Attributes

- GLuint **texture_id**

  Identificador de la textura cargada.
- GLenum **texture_unit**

  Unidad de textura a la que la textura está asignada.
- std::string **file_path**

  Ruta del archivo de la textura.

6.42.1 Detailed Description

Representa una textura en OpenGL.

La clase Texture permite la carga y manejo de texturas en OpenGL. Estas texturas pueden ser utilizadas en diferentes tipos de materiales y objetos 3D dentro de la escena. La clase gestiona el enlace y des-enlace de texturas, permitiendo su uso en shaders.

6.42.2 Constructor & Destructor Documentation

6.42.2.1 Texture()

```
Texture::Texture (
             const std::string & path,
             GLenum texture_unit,
             Texture_Type type = Texture_Type::COLOR)
```

Constructor que crea la textura a partir de un archivo.

Este constructor carga la textura desde una ruta de archivo específica. Se puede especificar el tipo de textura (por defecto es COLOR).

Parameters

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()

```
void udit::Texture::set_type (
             Texture_Type type)  [inline]
```

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()

```
Texture::Texture (
            const std::string & path,
            GLenum texture_unit,
            Texture_Type type = Texture_Type::COLOR)
```

Constructor que crea la textura a partir de un archivo.

Este constructor carga la textura desde una ruta de archivo específica. Se puede especificar el tipo de textura (por defecto es COLOR).

Parameters

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()

```
void udit::Texture::set_type (
             Texture_Type type)  [inline]
```

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_↩ CENTERED }

Public Member Functions

- **Window** (const std::string &title, int left_x, int top_y, unsigned width, unsigned height, const OpenGL_Context_Settings &context_details)
- Window (const char ∗title, int left_x, int top_y, unsigned width, unsigned height, const OpenGL_Context_Settings &context_details)

  Constructor de la ventana.
- ∼**Window** ()

  Destructor de la ventana.
- **Window** (const Window &)=delete
- Window & **operator=** (const Window &)=delete
- **Window** (Window &&other) noexcept
- Window & **operator=** (Window &&other) noexcept
- void **swap_buffers** ()

  Intercambiar los buffers de OpenGL.

6.44.1 Constructor & Destructor Documentation

6.44.1.1 Window()

```
udit::Window::Window (
            const char * title,
            int left_x,
            int top_y,
            unsigned width,
            unsigned height,
            const OpenGL_Context_Settings & context_details)
```

Constructor de la ventana.

Parameters

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_↩ CENTERED }

Public Member Functions

- **Window** (const std::string &title, int left_x, int top_y, unsigned width, unsigned height, const OpenGL_Context_Settings &context_details)
- Window (const char ∗title, int left_x, int top_y, unsigned width, unsigned height, const OpenGL_Context_Settings &context_details)

  Constructor de la ventana.
- **Window** (const Window &)=delete
- **Window** (Window &&other) noexcept
- ∼**Window** ()

  Destructor de la ventana.
- Window & **operator=** (const Window &)=delete
- Window & **operator=** (Window &&other) noexcept
- void **swap_buffers** ()

  Intercambiar los buffers de OpenGL.

6.45.1 Constructor & Destructor Documentation

6.45.1.1 Window()

```
udit::Window::Window (
             const char * title,
             int left_x,
             int top_y,
             unsigned width,
             unsigned height,
             const OpenGL_Context_Settings & context_details)
```

Constructor de la ventana.

Parameters

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

```cpp
//
//  Camera.hpp
//  GL_Scene
//
//  Created by Alonso García on 23/12/24.
//

#pragma once

#include "glm.hpp"
#include <gtc/matrix_transform.hpp>
#include <gtc/constants.hpp>

enum class CameraMovement
{
    FORWARD,
    BACKWARD,
    LEFT,
    RIGHT,
    UP,
    DOWN
};
class Camera
{
public:
    glm::vec3 position;

    glm::vec3 front;

    glm::vec3 up;

    glm::vec3 right;

    glm::vec3 world_up;

    float yaw;

    float pitch;

    float movement_speed;

    float mouse_sensitivity;

    float zoom;

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

    glm::mat4 get_view_matrix() const;

    void process_keyboard(CameraMovement direction, float delta_time);

    void process_mouse_movement(float x_offset, float y_offset, bool constraint_pitch = true);

private:
    void update_camera_vectors();
};
```

7.2 Cube.hpp

```
//
//  Cube.hpp
//  GL_Geometry
//
//  Created by Alonso García on 21/12/24.
//

#pragma once

#include "glad.h"

#include "Mesh.hpp"

namespace udit
{
    class Cube : public Mesh
    {
    private:
        float size;

    public:
        Cube();

        Cube(bool inverted);

        Cube(float size);

        Cube(float size, bool inverted);

    private:
        void create_cube(bool inverted = false);
    };
}
```

7.3 EventHandler.hpp

```
//
//  EventHandler.hpp
//  GL_Scene
//
//  Created by Alonso García on 23/12/24.
//

#pragma once

#include "SDL.h"
#include "glm.hpp"
#include "Camera.hpp"

class EventHandler
{
public:
    EventHandler(Camera& camera)
        : camera(camera), first_mouse(true), last_x(0.0f), last_y(0.0f) {}

    void handle_events(bool & running, float delta_time);

private:
    Camera & camera;

    bool first_mouse;

    float last_x;

    float last_y;

    void process_mouse_motion(const SDL_Event & event);

    void process_keyboard(const Uint8 * keystate, float delta_time);
};
```

7.4 GL_Scene/half.hpp File Reference

```
#include <utility>
#include <algorithm>
```

```
#include <istream>
#include <ostream>
#include <limits>
#include <stdexcept>
#include <climits>
#include <cmath>
#include <cstring>
#include <cstdlib>
```

Classes

- struct half_float::detail::conditional< bool, T, typename >

  Conditional type.
- struct half_float::detail::conditional< false, T, F >
- struct half_float::detail::bool_type< bool >

  Helper for tag dispatching.
- struct half_float::detail::is_float< typename >

  Type traits for floating-point types.
- struct half_float::detail::is_float< const T >
- struct half_float::detail::is_float< volatile T >
- struct half_float::detail::is_float< const volatile T >
- struct half_float::detail::is_float< float >
- struct half_float::detail::is_float< double >
- struct half_float::detail::is_float< long double >
- struct half_float::detail::bits< T >

  Type traits for floating-point bits.
- struct half_float::detail::bits< const T >
- struct half_float::detail::bits< volatile T >
- struct half_float::detail::bits< const volatile T >
- struct half_float::detail::bits< float >

  Unsigned integer of (at least) 32 bits width.
- struct half_float::detail::bits< double >

  Unsigned integer of (at least) 64 bits width.
- struct half_float::detail::binary_t

  Tag type for binary construction.
- struct half_float::detail::f31

  Class for 1.31 unsigned floating-point computation.
- class half_float::half
- struct half_float::detail::half_caster< T, U, R >
- struct half_float::detail::half_caster< half, U, R >
- struct half_float::detail::half_caster< T, half, R >
- struct half_float::detail::half_caster< half, half, R >
- class std::numeric_limits< half_float::half >

Namespaces

- namespace half_float
- namespace std

  Extensions to the C++ standard library.

Macros

- #define **HALF_GCC_VERSION** (__GNUC__*100+__GNUC_MINOR__)
- #define **HALF_ICC_VERSION** 0
- #define **HALF_ERRHANDLING** (HALF_ERRHANDLING_FLAGS||HALF_ERRHANDLING_ERRNO||HALF↩ _ERRHANDLING_FENV||HALF_ERRHANDLING_THROWS)
- #define **HALF_UNUSED_NOERR**(name)
- #define **HALF_CONSTEXPR**
- #define **HALF_CONSTEXPR_CONST** const
- #define **HALF_CONSTEXPR_NOERR**
- #define **HALF_NOEXCEPT**
- #define **HALF_NOTHROW** throw()
- #define **HALF_THREAD_LOCAL** static
- #define HALF_ENABLE_F16C_INTRINSICS __F16C__
- #define HALF_ERRHANDLING_OVERFLOW_TO_INEXACT 1
- #define HALF_ERRHANDLING_UNDERFLOW_TO_INEXACT 1
- #define HALF_ROUND_STYLE 1
- #define HUGE_VALH std::numeric_limits<half_float::half>::infinity()
- #define FP_FAST_FMAH 1
- #define HLF_ROUNDS HALF_ROUND_STYLE
- #define **FP_ILOGB0** INT_MIN
- #define **FP_ILOGBNAN** INT_MAX
- #define **FP_SUBNORMAL** 0
- #define **FP_ZERO** 1
- #define **FP_NAN** 2
- #define **FP_INFINITE** 3
- #define **FP_NORMAL** 4
- #define **FE_INVALID** 0x10
- #define **FE_DIVBYZERO** 0x08
- #define **FE_OVERFLOW** 0x04
- #define **FE_UNDERFLOW** 0x02
- #define **FE_INEXACT** 0x01
- #define **FE_ALL_EXCEPT** (FE_INVALID|FE_DIVBYZERO|FE_OVERFLOW|FE_UNDERFLOW|FE_↩ INEXACT)

Typedefs

- typedef bool_type< true > **half_float::detail::true_type**
- typedef bool_type< false > **half_float::detail::false_type**
- typedef unsigned short **half_float::detail::uint16**

  Unsigned integer of (at least) 16 bits width.
- typedef unsigned long **half_float::detail::uint32**

  Fastest unsigned integer of (at least) 32 bits width.
- typedef long **half_float::detail::int32**

  Fastest unsigned integer of (at least) 32 bits width.

Functions

Implementation defined classification and arithmetic

- template<typename T>
  bool half_float::detail::builtin_isinf (T arg)
- template<typename T>
  bool half_float::detail::builtin_isnan (T arg)
- template<typename T>
  bool half_float::detail::builtin_signbit (T arg)
- uint32 half_float::detail::sign_mask (uint32 arg)
- uint32 half_float::detail::arithmetic_shift (uint32 arg, int i)

Error handling

- int & half_float::detail::errflags ()
- void half_float::detail::raise (int HALF_UNUSED_NOERR(flags), bool HALF_UNUSED_NOERR(cond)=true)
- HALF_CONSTEXPR_NOERR bool half_float::detail::compsignal (unsigned int x, unsigned int y)
- HALF_CONSTEXPR_NOERR unsigned int half_float::detail::signal (unsigned int nan)
- HALF_CONSTEXPR_NOERR unsigned int half_float::detail::signal (unsigned int x, unsigned int y)
- HALF_CONSTEXPR_NOERR unsigned int half_float::detail::signal (unsigned int x, unsigned int y, unsigned int z)
- HALF_CONSTEXPR_NOERR unsigned int half_float::detail::select (unsigned int x, unsigned int HALF←_UNUSED_NOERR(y))
- HALF_CONSTEXPR_NOERR unsigned int half_float::detail::invalid ()
- HALF_CONSTEXPR_NOERR unsigned int half_float::detail::pole (unsigned int sign=0)
- HALF_CONSTEXPR_NOERR unsigned int half_float::detail::check_underflow (unsigned int arg)

Conversion and rounding

- template<std::float_round_style R>
  HALF_CONSTEXPR_NOERR unsigned int half_float::detail::overflow (unsigned int sign=0)
- template<std::float_round_style R>
  HALF_CONSTEXPR_NOERR unsigned int half_float::detail::underflow (unsigned int sign=0)
- template<std::float_round_style R, bool I>
  HALF_CONSTEXPR_NOERR unsigned int half_float::detail::rounded (unsigned int value, int g, int s)
- template<std::float_round_style R, bool E, bool I>
  unsigned int half_float::detail::integral (unsigned int value)
- template<std::float_round_style R, unsigned int F, bool S, bool N, bool I>
  unsigned int half_float::detail::fixed2half (uint32 m, int exp=14, unsigned int sign=0, int s=0)
- template<std::float_round_style R>
  unsigned int half_float::detail::float2half_impl (float value, true_type)
- template<std::float_round_style R>
  unsigned int half_float::detail::float2half_impl (double value, true_type)
- template<std::float_round_style R, typename T>
  unsigned int half_float::detail::float2half_impl (T value,...)
- template<std::float_round_style R, typename T>
  unsigned int half_float::detail::float2half (T value)
- template<std::float_round_style R, typename T>
  unsigned int half_float::detail::int2half (T value)
- float half_float::detail::half2float_impl (unsigned int value, float, true_type)
- double half_float::detail::half2float_impl (unsigned int value, double, true_type)
- template<typename T>
  T half_float::detail::half2float_impl (unsigned int value, T,...)
- template<typename T>
  T half_float::detail::half2float (unsigned int value)
- template<std::float_round_style R, bool E, bool I, typename T>
  T half_float::detail::half2int (unsigned int value)

Mathematics

- template<std::float_round_style R>
  uint32 half_float::detail::mulhi (uint32 x, uint32 y)
- uint32 half_float::detail::multiply64 (uint32 x, uint32 y)
- uint32 half_float::detail::divide64 (uint32 x, uint32 y, int &s)
- template<bool Q, bool R>
  unsigned int half_float::detail::mod (unsigned int x, unsigned int y, int ∗quo=NULL)
- template<unsigned int F>
  uint32 half_float::detail::sqrt (uint32 &r, int &exp)
- uint32 half_float::detail::exp2 (uint32 m, unsigned int n=32)
- uint32 half_float::detail::log2 (uint32 m, unsigned int n=32)
- std::pair< uint32, uint32 > half_float::detail::sincos (uint32 mz, unsigned int n=31)
- uint32 half_float::detail::atan2 (uint32 my, uint32 mx, unsigned int n=31)
- uint32 half_float::detail::angle_arg (unsigned int abs, int &k)
- std::pair< uint32, uint32 > half_float::detail::atan2_args (unsigned int abs)
- std::pair< uint32, uint32 > half_float::detail::hyperbolic_args (unsigned int abs, int &exp, unsigned int n=32)
- template<std::float_round_style R>
  unsigned int half_float::detail::exp2_post (uint32 m, int exp, bool esign, unsigned int sign=0, unsigned int n=32)
- template<std::float_round_style R, uint32 L>
  unsigned int half_float::detail::log2_post (uint32 m, int ilog, int exp, unsigned int sign=0)
- template<std::float_round_style R>
  unsigned int half_float::detail::hypot_post (uint32 r, int exp)
- template<std::float_round_style R>
  unsigned int half_float::detail::tangent_post (uint32 my, uint32 mx, int exp, unsigned int sign=0)
- template<std::float_round_style R, bool S>
  unsigned int half_float::detail::area (unsigned int arg)
- template<std::float_round_style R, bool C>
  unsigned int half_float::detail::erf (unsigned int arg)
- template<std::float_round_style R, bool L>
  unsigned int half_float::detail::gamma (unsigned int arg)

Comparison operators

- HALF_CONSTEXPR_NOERR bool half_float::operator== (half x, half y)
- HALF_CONSTEXPR_NOERR bool half_float::operator!= (half x, half y)
- HALF_CONSTEXPR_NOERR bool half_float::operator< (half x, half y)
- HALF_CONSTEXPR_NOERR bool half_float::operator> (half x, half y)
- HALF_CONSTEXPR_NOERR bool half_float::operator<= (half x, half y)
- HALF_CONSTEXPR_NOERR bool half_float::operator>= (half x, half y)

Arithmetic operators

- HALF_CONSTEXPR half half_float::operator+ (half arg)
- HALF_CONSTEXPR half half_float::operator- (half arg)
- half half_float::operator+ (half x, half y)
- half half_float::operator- (half x, half y)
- half half_float::operator∗ (half x, half y)
- half half_float::operator/ (half x, half y)

Input and output

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

Basic mathematical operations

- HALF_CONSTEXPR half half_float::fabs (half arg)
- HALF_CONSTEXPR half half_float::abs (half arg)
- half half_float::fmod (half x, half y)
- half half_float::remainder (half x, half y)
- half half_float::remquo (half x, half y, int ∗quo)
- half half_float::fma (half x, half y, half z)
- HALF_CONSTEXPR_NOERR half half_float::fmax (half x, half y)
- HALF_CONSTEXPR_NOERR half half_float::fmin (half x, half y)
- half half_float::fdim (half x, half y)
- half half_float::nanh (const char ∗arg)

Exponential functions

- half half_float::exp (half arg)
- half half_float::exp2 (half arg)
- half half_float::expm1 (half arg)
- half half_float::log (half arg)
- half half_float::log10 (half arg)
- half half_float::log2 (half arg)
- half half_float::log1p (half arg)

Power functions

- half half_float::sqrt (half arg)
- half half_float::rsqrt (half arg)
- half half_float::cbrt (half arg)
- half half_float::hypot (half x, half y)
- half half_float::hypot (half x, half y, half z)
- half half_float::pow (half x, half y)

Trigonometric functions

- void half_float::sincos (half arg, half ∗sin, half ∗cos)
- half half_float::sin (half arg)
- half half_float::cos (half arg)
- half half_float::tan (half arg)
- half half_float::asin (half arg)
- half half_float::acos (half arg)
- half half_float::atan (half arg)
- half half_float::atan2 (half y, half x)

Hyperbolic functions

- half half_float::sinh (half arg)
- half half_float::cosh (half arg)
- half half_float::tanh (half arg)
- half half_float::asinh (half arg)
- half half_float::acosh (half arg)
- half half_float::atanh (half arg)

Error and gamma functions

- half half_float::erf (half arg)
- half half_float::erfc (half arg)
- half half_float::lgamma (half arg)
- half half_float::tgamma (half arg)

Rounding

- half half_float::ceil (half arg)
- half half_float::floor (half arg)
- half half_float::trunc (half arg)
- half half_float::round (half arg)
- long half_float::lround (half arg)
- half half_float::rint (half arg)
- long half_float::lrint (half arg)
- half half_float::nearbyint (half arg)

Floating point manipulation

- half half_float::frexp (half arg, int ∗exp)
- half half_float::scalbln (half arg, long exp)
- half half_float::scalbn (half arg, int exp)
- half half_float::ldexp (half arg, int exp)
- half half_float::modf (half arg, half ∗iptr)
- int half_float::ilogb (half arg)
- half half_float::logb (half arg)
- half half_float::nextafter (half from, half to)
- half half_float::nexttoward (half from, long double to)
- HALF_CONSTEXPR half half_float::copysign (half x, half y)

Floating point classification

- HALF_CONSTEXPR int half_float::fpclassify (half arg)
- HALF_CONSTEXPR bool half_float::isfinite (half arg)
- HALF_CONSTEXPR bool half_float::isinf (half arg)
- HALF_CONSTEXPR bool half_float::isnan (half arg)
- HALF_CONSTEXPR bool half_float::isnormal (half arg)
- HALF_CONSTEXPR bool half_float::signbit (half arg)

Comparison

- HALF_CONSTEXPR bool half_float::isgreater (half x, half y)
- HALF_CONSTEXPR bool half_float::isgreaterequal (half x, half y)
- HALF_CONSTEXPR bool half_float::isless (half x, half y)
- HALF_CONSTEXPR bool half_float::islessequal (half x, half y)
- HALF_CONSTEXPR bool half_float::islessgreater (half x, half y)
- HALF_CONSTEXPR bool half_float::isunordered (half x, half y)

Casting

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

Error handling

- int half_float::feclearexcept (int excepts)
- int half_float::fetestexcept (int excepts)
- int half_float::feraiseexcept (int excepts)
- int half_float::fegetexceptflag (int ∗flagp, int excepts)
- int half_float::fesetexceptflag (const int ∗flagp, int excepts)
- void half_float::fethrowexcept (int excepts, const char ∗msg="")

Variables

- HALF_CONSTEXPR_CONST binary_t **half_float::detail::binary** = binary_t()

    Tag for binary construction.

7.4.1 Detailed Description

Main header file for half-precision functionality.

7.4.2 Macro Definition Documentation

7.4.2.1 FP_FAST_FMAH

```
#define FP_FAST_FMAH 1
```

Fast half-precision fma function. This symbol is defined if the fma() function generally executes as fast as, or faster than, a separate half-precision multiplication followed by an addition, which is always the case.

See also: Documentation for `FP_FAST_FMA`

7.4.2.2 HALF_ENABLE_F16C_INTRINSICS

```
#define HALF_ENABLE_F16C_INTRINSICS __F16C__
```

Enable F16C 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 `HUGE_VAL[F|L]` from `<cmath>` this symbol expands to a positive value signaling the overflow of an operation, in particular it just evaluates to positive infinity.

See also: Documentation for `HUGE_VAL`

7.4.3 Function Documentation

7.4.3.1 angle_arg()

```
uint32 half_float::detail::angle_arg (
             unsigned int abs,
             int & k)   [inline]
```

Reduce argument for trigonometric functions.

Parameters

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()

```
template<std::float_round_style R, bool S>
unsigned int half_float::detail::area (
             unsigned int arg)
```

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

Template Parameters

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

Parameters

arg	half-precision argument

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()

```
uint32 half_float::detail::arithmetic_shift (
            uint32 arg,
            int i)  [inline]
```

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()

```
uint32 half_float::detail::atan2 (
            uint32 my,
            uint32 mx,
            unsigned int n = 31)  [inline]
```

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

Parameters

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

Returns

arc tangent of *my* / *mx* as Q1.30

7.4.3.5 atan2_args()

```
std::pair< uint32, uint32 > half_float::detail::atan2_args (
            unsigned int abs)  [inline]
```

Get arguments for atan2 function.

Parameters

abs	half-precision floating-point value

Returns

abs and sqrt(1 - *abs*^*2*) as Q0.30

7.4.3.6 builtin_isinf()

```
template<typename T>
bool half_float::detail::builtin_isinf (
            T arg)
```

Check for infinity.

Template Parameters

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()

```
template<typename T>
bool half_float::detail::builtin_isnan (
            T arg)
```

Check for NaN.

Template Parameters

T	argument type (builtin floating-point type)

Parameters

arg	value to query

Return values

true	if not a number
false	else

7.4.3.8 builtin_signbit()

```
template<typename T>
bool half_float::detail::builtin_signbit (
            T arg)
```

Check sign.

Template Parameters

T	argument type (builtin floating-point type)

Parameters

arg	value to query

Return values

true	if signbit set
false	else

7.4.3.9 check_underflow()

```
HALF_CONSTEXPR_NOERR unsigned int half_float::detail::check_underflow (
            unsigned int arg) [inline]
```

Check value for underflow.

Parameters

arg	non-zero half-precision value to check

Returns

arg

Exceptions

FE_UNDERFLOW	if arg is subnormal

7.4.3.10 compsignal()

```
HALF_CONSTEXPR_NOERR bool half_float::detail::compsignal (
            unsigned int x,
            unsigned int y)  [inline]
```

Check and signal for any NaN.

Parameters

x	first half-precision value to check
y	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()

```
uint32 half_float::detail::divide64 (
            uint32 x,
            uint32 y,
            int & s)  [inline]
```

64-bit division.

Parameters

x	upper 32 bit of dividend
y	divisor
s	variable to store sticky bit for rounding

Returns

(*x* << 32) / *y*

7.4.3.12 erf()

```
template<std::float_round_style R, bool C>
unsigned int half_float::detail::erf (
            unsigned int arg)
```

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

Template Parameters

R	rounding mode to use
C	`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()

```
uint32 half_float::detail::exp2 (
            uint32 m,
            unsigned int n = 32)  [inline]
```

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

Parameters

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()

```
template<std::float_round_style R>
unsigned int half_float::detail::exp2_post (
             uint32 m,
             int exp,
             bool esign,
             unsigned int sign = 0,
             unsigned int n = 32)
```

Postprocessing for binary exponential.

Template Parameters

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 *I* is `true`

7.4.3.16 fixed2half()

```
template<std::float_round_style R, unsigned int F, bool S, bool N, bool I>
unsigned int half_float::detail::fixed2half (
            uint32 m,
            int exp = 14,
            unsigned int sign = 0,
            int s = 0)
```

Convert fixed point to half-precision floating-point.

Template Parameters

R	rounding mode to use
F	number of fractional bits in [11,31]
S	`true` for signed, `false` for unsigned
N	`true` for additional normalization step, `false` if already normalized to 1.F
I	`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 *I* is `true`

7.4.3.17 float2half()

```
template<std::float_round_style R, typename T>
unsigned int half_float::detail::float2half (
            T value)
```

Convert floating-point to half-precision.

Template Parameters

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]

```
template<std::float_round_style R>
unsigned int half_float::detail::float2half_impl (
            double value,
            true_type )
```

Convert IEEE double-precision to half-precision.

Template Parameters

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]

```
template<std::float_round_style R>
unsigned int half_float::detail::float2half_impl (
             float value,
             true_type )
```

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

Template Parameters

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]

```
template<std::float_round_style R, typename T>
unsigned int half_float::detail::float2half_impl (
             T value,
              ...)
```

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

Template Parameters

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

7.4.3.21 gamma()

```
template<std::float_round_style R, bool L>
unsigned int half_float::detail::gamma (
            unsigned int arg)
```

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

Template Parameters

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

Parameters

arg	half-precision floating-point value

Returns

lgamma/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()

```
template<typename T>
T half_float::detail::half2float (
             unsigned int value)
```

Convert half-precision to floating-point.

Template Parameters

T	type to convert to (builtin integer type)

Parameters

value	half-precision value to convert

Returns

floating-point value

7.4.3.23 half2float_impl() [1/3]

```
double half_float::detail::half2float_impl (
             unsigned int value,
             double ,
             true_type )  [inline]
```

Convert half-precision to IEEE double-precision.

Parameters

value	half-precision value to convert

Returns

double-precision value

7.4.3.24 half2float_impl() [2/3]

```
float half_float::detail::half2float_impl (
             unsigned int value,
             float ,
             true_type )  [inline]
```

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

Parameters

value	half-precision value to convert

Returns

single-precision value

7.4.3.25 half2float_impl() [3/3]

```
template<typename T>
T half_float::detail::half2float_impl (
             unsigned int value,
             T ,
              ...)
```

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

Template Parameters

T	type to convert to (builtin integer type)

Parameters

value	half-precision value to convert

Returns

floating-point value

7.4.3.26 half2int()

```
template<std::float_round_style R, bool E, bool I, typename T>
T half_float::detail::half2int (
            unsigned int value)
```

Convert half-precision floating-point to integer.

Template Parameters

R	rounding mode to use
E	`true` for round to even, `false` for round away from zero
I	`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

value	half-precision value to convert

Returns

rounded integer value

Exceptions

FE_INVALID	if value is not representable in type *T*
FE_INEXACT	if value had to be rounded and *I* 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
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()

```
template<std::float_round_style R>
unsigned int half_float::detail::hypot_post (
             uint32 r,
             int exp)
```

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()

```
template<std::float_round_style R, typename T>
unsigned int half_float::detail::int2half (
             T value)
```

Convert integer to half-precision floating-point.

Template Parameters

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

Parameters

value	integral value to convert

Returns

rounded half-precision value

Exceptions

FE_OVERFLOW	on overflows
FE_INEXACT	if value had to be rounded

7.4.3.30 integral()

```
template<std::float_round_style R, bool E, bool I>
unsigned int half_float::detail::integral (
            unsigned int value)
```

Round half-precision number to nearest integer value.

Template Parameters

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

Parameters

value	half-precision value 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 *I* is \`true\`

7.4.3.31 invalid()

```
HALF_CONSTEXPR_NOERR unsigned int half_float::detail::invalid ()  [inline]
```

Raise domain error and return NaN. return quiet NaN

Exceptions

FE_INVALID	

7.4.3.32 log2()

```
uint32 half_float::detail::log2 (
            uint32 m,
            unsigned int n = 32)  [inline]
```

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

Parameters

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()

```
template<std::float_round_style R, uint32 L>
unsigned int half_float::detail::log2_post (
            uint32 m,
            int ilog,
            int exp,
            unsigned int sign = 0)
```

Postprocessing for binary logarithm.

Template Parameters

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
y	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()

```
template<std::float_round_style R>
uint32 half_float::detail::mulhi (
             uint32 x,
             uint32 y)
```

upper part of 64-bit multiplication.

Template Parameters

R	rounding mode to use

Parameters

x	first factor
y	second factor

Returns

upper 32 bit of $x * y$

7.4.3.36 multiply64()

```
uint32 half_float::detail::multiply64 (
            uint32 x,
            uint32 y)  [inline]
```

64-bit multiplication.

Parameters

x	first factor
y	second factor

Returns

upper 32 bit of $x * y$ rounded to nearest

7.4.3.37 overflow()

```
template<std::float_round_style R>
HALF_CONSTEXPR_NOERR unsigned int half_float::detail::overflow (
            unsigned int sign = 0)
```

Half-precision overflow.

Template Parameters

R	rounding mode to use

Parameters

sign	half-precision value with sign bit only

Returns

rounded overflowing half-precision value

Exceptions

FE_OVERFLOW	

7.4.3.38 pole()

```
HALF_CONSTEXPR_NOERR unsigned int half_float::detail::pole (
            unsigned int sign = 0)  [inline]
```

Raise pole error and return infinity.

Parameters

sign	half-precision value with sign bit only

Returns

half-precision infinity with sign of *sign*

Exceptions

FE_DIVBYZERO	

7.4.3.39 raise()

```
void half_float::detail::raise (
             int  HALF_UNUSED_NOERRflags,
             bool  HALF_UNUSED_NOERRcond = true)  [inline]
```

Raise floating-point exception.

Parameters

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
I	`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 *I* is `true`

7.4.3.41 select()

```
HALF_CONSTEXPR_NOERR unsigned int half_float::detail::select (
            unsigned int x,
            unsigned int  HALF_UNUSED_NOERRy)  [inline]
```

Select value or signaling NaN.

Parameters

x	preferred half-precision value
y	ignored half-precision value except for signaling NaN

Returns

y if signaling NaN, *x* otherwise

Exceptions

FE_INVALID	if *y* is signaling NaN

7.4.3.42 sign_mask()

```
uint32 half_float::detail::sign_mask (
            uint32 arg)  [inline]
```

Platform-independent sign mask.

Parameters

arg	integer value in two's complement

Return values

-1	if *arg* negative
0	if *arg* positive

7.4.3.43 signal() [1/3]

```
HALF_CONSTEXPR_NOERR unsigned int half_float::detail::signal (
            unsigned int nan)  [inline]
```

Signal and silence signaling NaN.

Parameters

nan	half-precision NaN value

Returns

quiet NaN

Exceptions

FE_INVALID	if *nan* is signaling NaN

7.4.3.44 signal() [2/3]

```
HALF_CONSTEXPR_NOERR unsigned int half_float::detail::signal (
            unsigned int x,
            unsigned int y)  [inline]
```

Signal and silence signaling NaNs.

Parameters

x	first half-precision value to check
y	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]

```
HALF_CONSTEXPR_NOERR unsigned int half_float::detail::signal (
            unsigned int x,
            unsigned int y,
            unsigned int z)  [inline]
```

Signal and silence signaling NaNs.

Parameters

x	first half-precision value to check
y	second half-precision value to check
z	third half-precision value to check

Returns

quiet NaN

Exceptions

FE_INVALID	if *x*, *y* or *z* is signaling NaN

7.4.3.46 sincos()

```
std::pair< uint32, uint32 > half_float::detail::sincos (
            uint32 mz,
            unsigned int n = 31)  [inline]
```

Fixed point sine and cosine. This uses the CORDIC algorithm in rotation mode.

Parameters

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()

```
template<unsigned int F>
uint32 half_float::detail::sqrt (
            uint32 & r,
            int & exp)
```

Fixed point square root.

Template Parameters

F	number of fractional bits

Parameters

r	radicand in Q1.F fixed point format
exp	exponent

Returns

square root as Q1.F/2

7.4.3.48 tangent_post()

```
template<std::float_round_style R>
unsigned int half_float::detail::tangent_post (
            uint32 my,
            uint32 mx,
            int exp,
            unsigned int sign = 0)
```

Division and postprocessing for tangents.

Template Parameters

R	rounding mode to use

Parameters

my	dividend as Q1.31
mx	divisor as Q1.31
exp	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()

```
template<std::float_round_style R>
HALF_CONSTEXPR_NOERR unsigned int half_float::detail::underflow (
            unsigned int sign = 0)
```

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.

```
// half - IEEE 754-based half-precision floating-point library.
//
// Copyright (c) 2012-2021 Christian Rau <rauy@users.sourceforge.net>
//
// Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation
// files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy,
// modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the
// Software is furnished to do so, subject to the following conditions:
//
// The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
// WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
// COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
// ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

// Version 2.2.0


#ifndef HALF_HALF_HPP
#define HALF_HALF_HPP

#define HALF_GCC_VERSION (__GNUC__*100+__GNUC_MINOR__)

#if defined(__INTEL_COMPILER)
    #define HALF_ICC_VERSION __INTEL_COMPILER
#elif defined(__ICC)
    #define HALF_ICC_VERSION __ICC
#elif defined(__ICL)
    #define HALF_ICC_VERSION __ICL
#else
    #define HALF_ICC_VERSION 0
#endif

// check C++11 language features
#if defined(__clang__)                                      // clang
    #if __has_feature(cxx_static_assert) && !defined(HALF_ENABLE_CPP11_STATIC_ASSERT)
        #define HALF_ENABLE_CPP11_STATIC_ASSERT 1
    #endif
    #if __has_feature(cxx_constexpr) && !defined(HALF_ENABLE_CPP11_CONSTEXPR)
        #define HALF_ENABLE_CPP11_CONSTEXPR 1
    #endif
    #if __has_feature(cxx_noexcept) && !defined(HALF_ENABLE_CPP11_NOEXCEPT)
        #define HALF_ENABLE_CPP11_NOEXCEPT 1
    #endif
    #if __has_feature(cxx_user_literals) && !defined(HALF_ENABLE_CPP11_USER_LITERALS)
        #define HALF_ENABLE_CPP11_USER_LITERALS 1
    #endif
    #if __has_feature(cxx_thread_local) && !defined(HALF_ENABLE_CPP11_THREAD_LOCAL)
        #define HALF_ENABLE_CPP11_THREAD_LOCAL 1
    #endif
    #if (defined(__GXX_EXPERIMENTAL_CXX0X__) || __cplusplus >= 201103L) && !defined(HALF_ENABLE_CPP11_LONG_LONG)
        #define HALF_ENABLE_CPP11_LONG_LONG 1
    #endif
#elif HALF_ICC_VERSION && defined(__INTEL_CXX11_MODE__)     // Intel C++
    #if HALF_ICC_VERSION >= 1500 && !defined(HALF_ENABLE_CPP11_THREAD_LOCAL)
        #define HALF_ENABLE_CPP11_THREAD_LOCAL 1
    #endif
    #if HALF_ICC_VERSION >= 1500 && !defined(HALF_ENABLE_CPP11_USER_LITERALS)
        #define HALF_ENABLE_CPP11_USER_LITERALS 1
    #endif
    #if HALF_ICC_VERSION >= 1400 && !defined(HALF_ENABLE_CPP11_CONSTEXPR)
        #define HALF_ENABLE_CPP11_CONSTEXPR 1
    #endif
    #if HALF_ICC_VERSION >= 1400 && !defined(HALF_ENABLE_CPP11_NOEXCEPT)
        #define HALF_ENABLE_CPP11_NOEXCEPT 1
    #endif
    #if HALF_ICC_VERSION >= 1110 && !defined(HALF_ENABLE_CPP11_STATIC_ASSERT)
        #define HALF_ENABLE_CPP11_STATIC_ASSERT 1
    #endif
    #if HALF_ICC_VERSION >= 1110 && !defined(HALF_ENABLE_CPP11_LONG_LONG)
        #define HALF_ENABLE_CPP11_LONG_LONG 1
    #endif
```

```
#elif defined(__GNUC__)                                     // gcc
    #if defined(__GXX_EXPERIMENTAL_CXX0X__) || __cplusplus >= 201103L
        #if HALF_GCC_VERSION >= 408 && !defined(HALF_ENABLE_CPP11_THREAD_LOCAL)
            #define HALF_ENABLE_CPP11_THREAD_LOCAL 1
        #endif
        #if HALF_GCC_VERSION >= 407 && !defined(HALF_ENABLE_CPP11_USER_LITERALS)
            #define HALF_ENABLE_CPP11_USER_LITERALS 1
        #endif
        #if HALF_GCC_VERSION >= 406 && !defined(HALF_ENABLE_CPP11_CONSTEXPR)
            #define HALF_ENABLE_CPP11_CONSTEXPR 1
        #endif
        #if HALF_GCC_VERSION >= 406 && !defined(HALF_ENABLE_CPP11_NOEXCEPT)
            #define HALF_ENABLE_CPP11_NOEXCEPT 1
        #endif
        #if HALF_GCC_VERSION >= 403 && !defined(HALF_ENABLE_CPP11_STATIC_ASSERT)
            #define HALF_ENABLE_CPP11_STATIC_ASSERT 1
        #endif
        #if !defined(HALF_ENABLE_CPP11_LONG_LONG)
            #define HALF_ENABLE_CPP11_LONG_LONG 1
        #endif
    #endif
    #define HALF_TWOS_COMPLEMENT_INT 1
#elif defined(_MSC_VER)                                     // Visual C++
    #if _MSC_VER >= 1900 && !defined(HALF_ENABLE_CPP11_THREAD_LOCAL)
        #define HALF_ENABLE_CPP11_THREAD_LOCAL 1
    #endif
    #if _MSC_VER >= 1900 && !defined(HALF_ENABLE_CPP11_USER_LITERALS)
        #define HALF_ENABLE_CPP11_USER_LITERALS 1
    #endif
    #if _MSC_VER >= 1900 && !defined(HALF_ENABLE_CPP11_CONSTEXPR)
        #define HALF_ENABLE_CPP11_CONSTEXPR 1
    #endif
    #if _MSC_VER >= 1900 && !defined(HALF_ENABLE_CPP11_NOEXCEPT)
        #define HALF_ENABLE_CPP11_NOEXCEPT 1
    #endif
    #if _MSC_VER >= 1600 && !defined(HALF_ENABLE_CPP11_STATIC_ASSERT)
        #define HALF_ENABLE_CPP11_STATIC_ASSERT 1
    #endif
    #if _MSC_VER >= 1310 && !defined(HALF_ENABLE_CPP11_LONG_LONG)
        #define HALF_ENABLE_CPP11_LONG_LONG 1
    #endif
    #define HALF_TWOS_COMPLEMENT_INT 1
    #define HALF_POP_WARNINGS 1
    #pragma warning(push)
    #pragma warning(disable : 4099 4127 4146)   //struct vs class, constant in if, negative unsigned
#endif

// check C++11 library features
#include <utility>
#if defined(_LIBCPP_VERSION)                                // libc++
    #if defined(__GXX_EXPERIMENTAL_CXX0X__) || __cplusplus >= 201103
        #ifndef HALF_ENABLE_CPP11_TYPE_TRAITS
            #define HALF_ENABLE_CPP11_TYPE_TRAITS 1
        #endif
        #ifndef HALF_ENABLE_CPP11_CSTDINT
            #define HALF_ENABLE_CPP11_CSTDINT 1
        #endif
        #ifndef HALF_ENABLE_CPP11_CMATH
            #define HALF_ENABLE_CPP11_CMATH 1
        #endif
        #ifndef HALF_ENABLE_CPP11_HASH
            #define HALF_ENABLE_CPP11_HASH 1
        #endif
        #ifndef HALF_ENABLE_CPP11_CFENV
            #define HALF_ENABLE_CPP11_CFENV 1
        #endif
    #endif
#elif defined(__GLIBCXX__)                                  // libstdc++
    #if defined(__GXX_EXPERIMENTAL_CXX0X__) || __cplusplus >= 201103
        #ifdef __clang__
            #if __GLIBCXX__ >= 20080606 && !defined(HALF_ENABLE_CPP11_TYPE_TRAITS)
                #define HALF_ENABLE_CPP11_TYPE_TRAITS 1
            #endif
            #if __GLIBCXX__ >= 20080606 && !defined(HALF_ENABLE_CPP11_CSTDINT)
                #define HALF_ENABLE_CPP11_CSTDINT 1
            #endif
            #if __GLIBCXX__ >= 20080606 && !defined(HALF_ENABLE_CPP11_CMATH)
                #define HALF_ENABLE_CPP11_CMATH 1
            #endif
            #if __GLIBCXX__ >= 20080606 && !defined(HALF_ENABLE_CPP11_HASH)
                #define HALF_ENABLE_CPP11_HASH 1
            #endif
            #if __GLIBCXX__ >= 20080606 && !defined(HALF_ENABLE_CPP11_CFENV)
                #define HALF_ENABLE_CPP11_CFENV 1
            #endif
        #else
            #if HALF_GCC_VERSION >= 403 && !defined(HALF_ENABLE_CPP11_TYPE_TRAITS)
```

```
                #define HALF_ENABLE_CPP11_TYPE_TRAITS 1
            #endif
            #if HALF_GCC_VERSION >= 403 && !defined(HALF_ENABLE_CPP11_CSTDINT)
                #define HALF_ENABLE_CPP11_CSTDINT 1
            #endif
            #if HALF_GCC_VERSION >= 403 && !defined(HALF_ENABLE_CPP11_CMATH)
                #define HALF_ENABLE_CPP11_CMATH 1
            #endif
            #if HALF_GCC_VERSION >= 403 && !defined(HALF_ENABLE_CPP11_HASH)
                #define HALF_ENABLE_CPP11_HASH 1
            #endif
            #if HALF_GCC_VERSION >= 403 && !defined(HALF_ENABLE_CPP11_CFENV)
                #define HALF_ENABLE_CPP11_CFENV 1
            #endif
        #endif
    #endif
#elif defined(_CPPLIB_VER)                                  // Dinkumware/Visual C++
    #if _CPPLIB_VER >= 520 && !defined(HALF_ENABLE_CPP11_TYPE_TRAITS)
        #define HALF_ENABLE_CPP11_TYPE_TRAITS 1
    #endif
    #if _CPPLIB_VER >= 520 && !defined(HALF_ENABLE_CPP11_CSTDINT)
            #define HALF_ENABLE_CPP11_CSTDINT 1
    #endif
    #if _CPPLIB_VER >= 520 && !defined(HALF_ENABLE_CPP11_HASH)
        #define HALF_ENABLE_CPP11_HASH 1
    #endif
    #if _CPPLIB_VER >= 610 && !defined(HALF_ENABLE_CPP11_CMATH)
        #define HALF_ENABLE_CPP11_CMATH 1
    #endif
    #if _CPPLIB_VER >= 610 && !defined(HALF_ENABLE_CPP11_CFENV)
        #define HALF_ENABLE_CPP11_CFENV 1
    #endif
#endif
#undef HALF_GCC_VERSION
#undef HALF_ICC_VERSION

// any error throwing C++ exceptions?
#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)
#define HALF_ERRHANDLING_THROWS 1
#endif

// any error handling enabled?
#define HALF_ERRHANDLING (HALF_ERRHANDLING_FLAGS||HALF_ERRHANDLING_ERRNO||HALF_ERRHANDLING_FENV||HALF_ERRHANDLING_THROWS)

#if HALF_ERRHANDLING
    #define HALF_UNUSED_NOERR(name) name
#else
    #define HALF_UNUSED_NOERR(name)
#endif

// support constexpr
#if HALF_ENABLE_CPP11_CONSTEXPR
    #define HALF_CONSTEXPR              constexpr
    #define HALF_CONSTEXPR_CONST        constexpr
    #if HALF_ERRHANDLING
        #define HALF_CONSTEXPR_NOERR
    #else
        #define HALF_CONSTEXPR_NOERR    constexpr
    #endif
#else
    #define HALF_CONSTEXPR
    #define HALF_CONSTEXPR_CONST        const
    #define HALF_CONSTEXPR_NOERR
#endif

// support noexcept
#if HALF_ENABLE_CPP11_NOEXCEPT
    #define HALF_NOEXCEPT   noexcept
    #define HALF_NOTHROW    noexcept
#else
    #define HALF_NOEXCEPT
    #define HALF_NOTHROW    throw()
#endif

// support thread storage
#if HALF_ENABLE_CPP11_THREAD_LOCAL
    #define HALF_THREAD_LOCAL   thread_local
#else
    #define HALF_THREAD_LOCAL   static
#endif

#include <utility>
#include <algorithm>
#include <istream>
```

```
#include <ostream>
#include <limits>
#include <stdexcept>
#include <climits>
#include <cmath>
#include <cstring>
#include <cstdlib>
#if HALF_ENABLE_CPP11_TYPE_TRAITS
    #include <type_traits>
#endif
#if HALF_ENABLE_CPP11_CSTDINT
    #include <cstdint>
#endif
#if HALF_ERRHANDLING_ERRNO
    #include <cerrno>
#endif
#if HALF_ENABLE_CPP11_CFENV
    #include <cfenv>
#endif
#if HALF_ENABLE_CPP11_HASH
    #include <functional>
#endif


#ifndef HALF_ENABLE_F16C_INTRINSICS
    #define HALF_ENABLE_F16C_INTRINSICS __F16C__
#endif
#if HALF_ENABLE_F16C_INTRINSICS
    #include <immintrin.h>
#endif

#ifdef HALF_DOXYGEN_ONLY
#define HALF_ARITHMETIC_TYPE (undefined)

#define HALF_ERRHANDLING_FLAGS  0

#define HALF_ERRHANDLING_ERRNO  0

#define HALF_ERRHANDLING_FENV   0

#define HALF_ERRHANDLING_THROW_INVALID      (undefined)

#define HALF_ERRHANDLING_THROW_DIVBYZERO    (undefined)

#define HALF_ERRHANDLING_THROW_OVERFLOW     (undefined)

#define HALF_ERRHANDLING_THROW_UNDERFLOW    (undefined)

#define HALF_ERRHANDLING_THROW_INEXACT      (undefined)
#endif

#ifndef HALF_ERRHANDLING_OVERFLOW_TO_INEXACT
#define HALF_ERRHANDLING_OVERFLOW_TO_INEXACT    1
#endif

#ifndef HALF_ERRHANDLING_UNDERFLOW_TO_INEXACT
#define HALF_ERRHANDLING_UNDERFLOW_TO_INEXACT   1
#endif

#ifndef HALF_ROUND_STYLE
    #define HALF_ROUND_STYLE    1       // = std::round_to_nearest
#endif

#define HUGE_VALH   std::numeric_limits<half_float::half>::infinity()

#define FP_FAST_FMAH    1

#define HLF_ROUNDS  HALF_ROUND_STYLE

#ifndef FP_ILOGB0
    #define FP_ILOGB0       INT_MIN
#endif
#ifndef FP_ILOGBNAN
    #define FP_ILOGBNAN     INT_MAX
#endif
#ifndef FP_SUBNORMAL
    #define FP_SUBNORMAL    0
#endif
#ifndef FP_ZERO
    #define FP_ZERO         1
#endif
#ifndef FP_NAN
    #define FP_NAN          2
#endif
#ifndef FP_INFINITE
    #define FP_INFINITE     3
#endif
```

```
#ifndef FP_NORMAL
    #define FP_NORMAL       4
#endif

#if !HALF_ENABLE_CPP11_CFENV && !defined(FE_ALL_EXCEPT)
    #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)
#endif


namespace half_float
{
    class half;

#if HALF_ENABLE_CPP11_USER_LITERALS
    namespace literal
    {
        half operator "" _h(long double);
    }
#endif

    namespace detail
    {
    #if HALF_ENABLE_CPP11_TYPE_TRAITS
        template<bool B,typename T,typename F> struct conditional : std::conditional<B,T,F> {};

        template<bool B> struct bool_type : std::integral_constant<bool,B> {};
        using std::true_type;
        using std::false_type;

        template<typename T> struct is_float : std::is_floating_point<T> {};
    #else
        template<bool,typename T,typename> struct conditional { typedef T type; };
        template<typename T,typename F> struct conditional<false,T,F> { typedef F type; };

        template<bool> struct bool_type {};
        typedef bool_type<true> true_type;
        typedef bool_type<false> false_type;

        template<typename> struct is_float : false_type {};
        template<typename T> struct is_float<const T> : is_float<T> {};
        template<typename T> struct is_float<volatile T> : is_float<T> {};
        template<typename T> struct is_float<const volatile T> : is_float<T> {};
        template<> struct is_float<float> : true_type {};
        template<> struct is_float<double> : true_type {};
        template<> struct is_float<long double> : true_type {};
    #endif

        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> {};
        template<typename T> struct bits<const volatile T> : bits<T> {};

    #if HALF_ENABLE_CPP11_CSTDINT
        typedef std::uint_least16_t uint16;

        typedef std::uint_fast32_t uint32;

        typedef std::int_fast32_t int32;

        template<> struct bits<float> { typedef std::uint_least32_t type; };

        template<> struct bits<double> { typedef std::uint_least64_t type; };
    #else
        typedef unsigned short uint16;

        typedef unsigned long uint32;

        typedef long int32;

        template<> struct bits<float> : conditional<std::numeric_limits<unsigned int>::digits>=32,unsigned int,unsigned long> {};

        #if HALF_ENABLE_CPP11_LONG_LONG
            template<> struct bits<double> : conditional<std::numeric_limits<unsigned long>::digits>=64,unsigned long,unsigned long long> {};
        #else
            template<> struct bits<double> { typedef unsigned long type; };
        #endif
    #endif

    #ifdef HALF_ARITHMETIC_TYPE
        typedef HALF_ARITHMETIC_TYPE internal_t;
```

```
#endif

    struct binary_t {};

    HALF_CONSTEXPR_CONST binary_t binary = binary_t();


    template<typename T> bool builtin_isinf(T arg)
    {
    #if HALF_ENABLE_CPP11_CMATH
        return std::isinf(arg);
    #elif defined(_MSC_VER)
        return !::_finite(static_cast<double>(arg)) && !::_isnan(static_cast<double>(arg));
    #else
        return arg == std::numeric_limits<T>::infinity() || arg == -std::numeric_limits<T>::infinity();
    #endif
    }

    template<typename T> bool builtin_isnan(T arg)
    {
    #if HALF_ENABLE_CPP11_CMATH
        return std::isnan(arg);
    #elif defined(_MSC_VER)
        return ::_isnan(static_cast<double>(arg)) != 0;
    #else
        return arg != arg;
    #endif
    }

    template<typename T> bool builtin_signbit(T arg)
    {
    #if HALF_ENABLE_CPP11_CMATH
        return std::signbit(arg);
    #else
        return arg < T() || (arg == T() && T(1)/arg < T());
    #endif
    }

    inline uint32 sign_mask(uint32 arg)
    {
        static const int N = std::numeric_limits<uint32>::digits - 1;
    #if HALF_TWOS_COMPLEMENT_INT
        return static_cast<int32>(arg) » N;
    #else
        return -((arg»N)&1);
    #endif
    }

    inline uint32 arithmetic_shift(uint32 arg, int i)
    {
    #if HALF_TWOS_COMPLEMENT_INT
        return static_cast<int32>(arg) » i;
    #else
        return static_cast<int32>(arg)/(static_cast<int32>(1)«i) - ((arg»(std::numeric_limits<uint32>::digits-1))&1);
    #endif
    }


    inline int& errflags() { HALF_THREAD_LOCAL int flags = 0; return flags; }

    inline void raise(int HALF_UNUSED_NOERR(flags), bool HALF_UNUSED_NOERR(cond) = true)
    {
    #if HALF_ERRHANDLING
        if(!cond)
            return;
    #if HALF_ERRHANDLING_FLAGS
        errflags() |= flags;
    #endif
    #if HALF_ERRHANDLING_ERRNO
        if(flags & FE_INVALID)
            errno = EDOM;
        else if(flags & (FE_DIVBYZERO|FE_OVERFLOW|FE_UNDERFLOW))
            errno = ERANGE;
    #endif
    #if HALF_ERRHANDLING_FENV && HALF_ENABLE_CPP11_CFENV
        std::feraiseexcept(flags);
    #endif
    #ifdef HALF_ERRHANDLING_THROW_INVALID
        if(flags & FE_INVALID)
            throw std::domain_error(HALF_ERRHANDLING_THROW_INVALID);
    #endif
    #ifdef HALF_ERRHANDLING_THROW_DIVBYZERO
        if(flags & FE_DIVBYZERO)
            throw std::domain_error(HALF_ERRHANDLING_THROW_DIVBYZERO);
    #endif
```

```
        #ifdef HALF_ERRHANDLING_THROW_OVERFLOW
            if(flags & FE_OVERFLOW)
                throw std::overflow_error(HALF_ERRHANDLING_THROW_OVERFLOW);
        #endif
        #ifdef HALF_ERRHANDLING_THROW_UNDERFLOW
            if(flags & FE_UNDERFLOW)
                throw std::underflow_error(HALF_ERRHANDLING_THROW_UNDERFLOW);
        #endif
        #ifdef HALF_ERRHANDLING_THROW_INEXACT
            if(flags & FE_INEXACT)
                throw std::range_error(HALF_ERRHANDLING_THROW_INEXACT);
        #endif
        #if HALF_ERRHANDLING_UNDERFLOW_TO_INEXACT
            if((flags & FE_UNDERFLOW) && !(flags & FE_INEXACT))
                raise(FE_INEXACT);
        #endif
        #if HALF_ERRHANDLING_OVERFLOW_TO_INEXACT
            if((flags & FE_OVERFLOW) && !(flags & FE_INEXACT))
                raise(FE_INEXACT);
        #endif
        #endif
        }

        inline HALF_CONSTEXPR_NOERR bool compsignal(unsigned int x, unsigned int y)
        {
        #if HALF_ERRHANDLING
            raise(FE_INVALID, (x&0x7FFF)>0x7C00 || (y&0x7FFF)>0x7C00);
        #endif
            return (x&0x7FFF) > 0x7C00 || (y&0x7FFF) > 0x7C00;
        }

        inline HALF_CONSTEXPR_NOERR unsigned int signal(unsigned int nan)
        {
        #if HALF_ERRHANDLING
            raise(FE_INVALID, !(nan&0x200));
        #endif
            return nan | 0x200;
        }

        inline HALF_CONSTEXPR_NOERR unsigned int signal(unsigned int x, unsigned int y)
        {
        #if HALF_ERRHANDLING
            raise(FE_INVALID, ((x&0x7FFF)>0x7C00 && !(x&0x200)) || ((y&0x7FFF)>0x7C00 && !(y&0x200)));
        #endif
            return ((x&0x7FFF)>0x7C00) ? (x|0x200) : (y|0x200);
        }

        inline HALF_CONSTEXPR_NOERR unsigned int signal(unsigned int x, unsigned int y, unsigned int z)
        {
        #if HALF_ERRHANDLING
            raise(FE_INVALID, ((x&0x7FFF)>0x7C00 && !(x&0x200)) || ((y&0x7FFF)>0x7C00 && !(y&0x200)) || ((z&0x7FFF)>0x7C00 && !(z&0x200)));
        #endif
            return ((x&0x7FFF)>0x7C00) ? (x|0x200) : ((y&0x7FFF)>0x7C00) ? (y|0x200) : (z|0x200);
        }

        inline HALF_CONSTEXPR_NOERR unsigned int select(unsigned int x, unsigned int HALF_UNUSED_NOERR(y))
        {
        #if HALF_ERRHANDLING
            return (((y&0x7FFF)>0x7C00) && !(y&0x200)) ? signal(y) : x;
        #else
            return x;
        #endif
        }

        inline HALF_CONSTEXPR_NOERR unsigned int invalid()
        {
        #if HALF_ERRHANDLING
            raise(FE_INVALID);
        #endif
            return 0x7FFF;
        }

        inline HALF_CONSTEXPR_NOERR unsigned int pole(unsigned int sign = 0)
        {
        #if HALF_ERRHANDLING
            raise(FE_DIVBYZERO);
        #endif
            return sign | 0x7C00;
        }

        inline HALF_CONSTEXPR_NOERR unsigned int check_underflow(unsigned int arg)
        {
        #if HALF_ERRHANDLING && !HALF_ERRHANDLING_UNDERFLOW_TO_INEXACT
            raise(FE_UNDERFLOW, !(arg&0x7C00));
        #endif
            return arg;
        }


        template<std::float_round_style R> HALF_CONSTEXPR_NOERR unsigned int overflow(unsigned int sign = 0)
        {
        #if HALF_ERRHANDLING
            raise(FE_OVERFLOW);
        #endif
            return  (R==std::round_toward_infinity) ? (sign+0x7C00-(sign»15)) :
                    (R==std::round_toward_neg_infinity) ? (sign+0x7BFF+(sign»15)) :
                    (R==std::round_toward_zero) ? (sign|0x7BFF) :
                    (sign|0x7C00);
        }

        template<std::float_round_style R> HALF_CONSTEXPR_NOERR unsigned int underflow(unsigned int sign = 0)
        {
        #if HALF_ERRHANDLING
            raise(FE_UNDERFLOW);
        #endif
            return  (R==std::round_toward_infinity) ? (sign+1-(sign»15)) :
                    (R==std::round_toward_neg_infinity) ? (sign+(sign»15)) :
                    sign;
        }

        template<std::float_round_style R,bool I> HALF_CONSTEXPR_NOERR unsigned int rounded(unsigned int value, int g, int s)
        {
        #if HALF_ERRHANDLING
            value +=    (R==std::round_to_nearest) ? (g&(s|value)) :
                        (R==std::round_toward_infinity) ? (~(value»15)&(g|s)) :
                        (R==std::round_toward_neg_infinity) ? ((value»15)&(g|s)) : 0;
            if((value&0x7C00) == 0x7C00)
                raise(FE_OVERFLOW);
            else if(value & 0x7C00)
                raise(FE_INEXACT, I || (g|s)!=0);
            else
                raise(FE_UNDERFLOW, !(HALF_ERRHANDLING_UNDERFLOW_TO_INEXACT) || I || (g|s)!=0);
            return value;
        #else
            return  (R==std::round_to_nearest) ? (value+(g&(s|value))) :
                    (R==std::round_toward_infinity) ? (value+(~(value»15)&(g|s))) :
                    (R==std::round_toward_neg_infinity) ? (value+((value»15)&(g|s))) :
                    value;
        #endif
        }

        template<std::float_round_style R,bool E,bool I> unsigned int integral(unsigned int value)
        {
            unsigned int abs = value & 0x7FFF;
            if(abs < 0x3C00)
            {
                raise(FE_INEXACT, I);
                return ((R==std::round_to_nearest) ? (0x3C00&-static_cast<unsigned>(abs>=(0x3800+E))) :
                        (R==std::round_toward_infinity) ? (0x3C00&-(~(value»15)&(abs!=0))) :
                        (R==std::round_toward_neg_infinity) ? (0x3C00&-static_cast<unsigned>(value>0x8000)) :
                        0) | (value&0x8000);
            }
            if(abs >= 0x6400)
                return (abs>0x7C00) ? signal(value) : value;
            unsigned int exp = 25 - (abs»10), mask = (1«exp) - 1;
            raise(FE_INEXACT, I && (value&mask));
            return ((    (R==std::round_to_nearest) ? ((1«(exp-1))-(~(value»exp)&E)) :
                        (R==std::round_toward_infinity) ? (mask&((value»15)-1)) :
                        (R==std::round_toward_neg_infinity) ? (mask&-(value»15)) :
                        0) + value) & ~mask;
        }

        template<std::float_round_style R,unsigned int F,bool S,bool N,bool I> unsigned int fixed2half(uint32 m, int exp = 14, unsigned int sign = 0, int s = 0)
        {
            if(S)
            {
                uint32 msign = sign_mask(m);
                m = (m^msign) - msign;
                sign = msign & 0x8000;
            }
            if(N)
                for(; m<(static_cast<uint32>(1)«F) && exp; m«=1,--exp) ;
            else if(exp < 0)
                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));
            return rounded<R,I>(sign+(exp«10)+(m»(F-10)), (m»(F-11))&1, s|((m&((static_cast<uint32>(1)«(F-11))-1))!=0));
        }

        template<std::float_round_style R> unsigned int float2half_impl(float value, true_type)
        {
        #if HALF_ENABLE_F16C_INTRINSICS
            return _mm_cvtsi128_si32(_mm_cvtps_ph(_mm_set_ss(value),
                (R==std::round_to_nearest) ? _MM_FROUND_TO_NEAREST_INT :
                (R==std::round_toward_zero) ? _MM_FROUND_TO_ZERO :
                (R==std::round_toward_infinity) ? _MM_FROUND_TO_POS_INF :
                (R==std::round_toward_neg_infinity) ? _MM_FROUND_TO_NEG_INF :
                _MM_FROUND_CUR_DIRECTION));
        #else
            bits<float>::type fbits;
            std::memcpy(&fbits, &value, sizeof(float));
        #if 1
            unsigned int sign = (fbits»16) & 0x8000;
            fbits &= 0x7FFFFFFF;
            if(fbits >= 0x7F800000)
                return sign | 0x7C00 | ((fbits>0x7F800000) ? (0x200|((fbits»13)&0x3FF)) : 0);
            if(fbits >= 0x47800000)
                return overflow<R>(sign);
            if(fbits >= 0x38800000)
                return rounded<R,false>(sign|(((fbits»23)-112)«10)|((fbits»13)&0x3FF), (fbits»12)&1, (fbits&0xFFF)!=0);
            if(fbits >= 0x33000000)
            {
                int i = 125 - (fbits»23);
                fbits = (fbits&0x7FFFFF) | 0x800000;
                return rounded<R,false>(sign|(fbits»(i+1)), (fbits»i)&1, (fbits&((static_cast<uint32>(1)«i)-1))!=0);
            }
            if(fbits != 0)
                return underflow<R>(sign);
            return sign;
        #else
            static const uint16 base_table[512] = {
                0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
                0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
                0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
                0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
                0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
                0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
                0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0002, 0x0004, 0x0008, 0x0010, 0x0020, 0x0040, 0x0080, 0x0100,
                0x0200, 0x0400, 0x0800, 0x0C00, 0x1000, 0x1400, 0x1800, 0x1C00, 0x2000, 0x2400, 0x2800, 0x2C00, 0x3000, 0x3400, 0x3800, 0x3C00,
                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, 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, 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, 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, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7BFF, 0x7C00,
                0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000,
                0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000,
                0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000,
                0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000,
                0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000,
                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, 0x8008, 0x8010, 0x8020, 0x8040, 0x8080, 0x8100,
                0x8200, 0x8400, 0x8800, 0x8C00, 0x9000, 0x9400, 0x9800, 0x9C00, 0xA000, 0xA400, 0xA800, 0xAC00, 0xB000, 0xB400, 0xB800, 0xBC00,
                0xC000, 0xC400, 0xC800, 0xCC00, 0xD000, 0xD400, 0xD800, 0xDC00, 0xE000, 0xE400,
0xE800, 0xEC00, 0xF000, 0xF400, 0xF800, 0xFBFF,
                0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF,
0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF,
                0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF,
0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF,
                0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF,
0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF,
                0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF,
0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF,
                0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF,
0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF,
                0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF,
0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF,
                0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF,
0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFBFF, 0xFC00 };
            static const unsigned char shift_table[256] = {
                24, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25,
25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25,
                25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25,
25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25,
                25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25,
25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25,
                25, 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,
                13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 24, 24, 24, 24, 24, 24,
24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24,
                24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24,
24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24,
                24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24,
24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24,
                24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24,
24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 13 };
            int sexp = fbits » 23, exp = sexp & 0xFF, i = shift_table[exp];
            fbits &= 0x7FFFFF;
            uint32 m = (fbits|((exp!=0)«23)) & -static_cast<uint32>(exp!=0xFF);
            return rounded<R,false>(base_table[sexp]+(fbits»i), (m»(i-1))&1,
(((static_cast<uint32>(1)«(i-1))-1)&m)!=0);
        #endif
        #endif
        }

        template<std::float_round_style R> unsigned int float2half_impl(double value, true_type)
        {
        #if HALF_ENABLE_F16C_INTRINSICS
            if(R == std::round_indeterminate)
                return _mm_cvtsi128_si32(_mm_cvtps_ph(_mm_cvtpd_ps(_mm_set_sd(value)),
_MM_FROUND_CUR_DIRECTION));
        #endif
            bits<double>::type dbits;
            std::memcpy(&dbits, &value, sizeof(double));
            uint32 hi = dbits » 32, lo = dbits & 0xFFFFFFFF;
            unsigned int sign = (hi»16) & 0x8000;
            hi &= 0x7FFFFFFF;
            if(hi >= 0x7FF00000)
                return sign | 0x7C00 | ((dbits&0xFFFFFFFFFFFFF) ? (0x200|((hi»10)&0x3FF)) : 0);
            if(hi >= 0x40F00000)
                return overflow<R>(sign);
            if(hi >= 0x3F100000)
                return rounded<R,false>(sign|(((hi»20)-1008)«10)|((hi»10)&0x3FF), (hi»9)&1,
((hi&0x1FF)|lo)!=0);
            if(hi >= 0x3E600000)
            {
                int i = 1018 - (hi»20);
                hi = (hi&0xFFFFF) | 0x100000;
                return rounded<R,false>(sign|(hi»(i+1)), (hi»i)&1,
((hi&((static_cast<uint32>(1)«i)-1))|lo)!=0);
            }
            if((hi|lo) != 0)
                return underflow<R>(sign);
            return sign;
        }

        template<std::float_round_style R,typename T> unsigned int float2half_impl(T value, ...)
        {
            unsigned int hbits = static_cast<unsigned>(builtin_signbit(value)) « 15;
            if(value == T())
                return hbits;
            if(builtin_isnan(value))
                return hbits | 0x7FFF;
            if(builtin_isinf(value))
                return hbits | 0x7C00;
            int exp;
            std::frexp(value, &exp);
            if(exp > 16)
                return overflow<R>(hbits);
            if(exp < -13)
                value = std::ldexp(value, 25);
            else
            {
                value = std::ldexp(value, 12-exp);
                hbits |= ((exp+13)«10);
            }
            T ival, frac = std::modf(value, &ival);
            int m = std::abs(static_cast<int>(ival));
            return rounded<R,false>(hbits+(m»1), m&1, frac!=T());
        }

        template<std::float_round_style R,typename T> unsigned int float2half(T value)
        {
            return float2half_impl<R>(value, bool_type<std::numeric_limits<T>::is_iec559&&sizeof(typename bits<T>::type)==sizeof(T)>());
        }

        template<std::float_round_style R,typename T> unsigned int int2half(T value)
        {
            unsigned int bits = static_cast<unsigned>(value<0) « 15;
            if(!value)
                return bits;
            if(bits)
                value = -value;
            if(value > 0xFFFF)
                return overflow<R>(bits);
            unsigned int m = static_cast<unsigned int>(value), exp = 24;
            for(; m<0x400; m«=1,--exp) ;
            for(; m>0x7FF; m»=1,++exp) ;
            bits |= (exp«10) + m;
            return (exp>24) ? rounded<R,false>(bits, (value»(exp-25))&1, (((1«(exp-25))-1)&value)!=0) : bits;
        }

        inline float half2float_impl(unsigned int value, float, true_type)
        {
        #if HALF_ENABLE_F16C_INTRINSICS
            return _mm_cvtss_f32(_mm_cvtph_ps(_mm_cvtsi32_si128(value)));
        #else
        #if 0
            bits<float>::type fbits = static_cast<bits<float>::type>(value&0x8000) « 16;
            int abs = value & 0x7FFF;
            if(abs)
            {
                fbits |= 0x38000000 « static_cast<unsigned>(abs>=0x7C00);
                for(; abs<0x400; abs«=1,fbits-=0x800000) ;
                fbits += static_cast<bits<float>::type>(abs) « 13;
            }
        #else
            static const bits<float>::type mantissa_table[2048] = {
                0x00000000, 0x33800000, 0x34000000, 0x34400000, 0x34800000, 0x34A00000, 0x34C00000, 0x34E00000, 0x35000000, 0x35100000, 0x35200000, 0x35300000, 0x35400000, 0x35500000, 0x35600000, 0x35700000,
                0x35800000, 0x35880000, 0x35900000, 0x35980000, 0x35A00000, 0x35A80000, 0x35B00000, 0x35B80000, 0x35C00000, 0x35C80000, 0x35D00000, 0x35D80000, 0x35E00000, 0x35E80000, 0x35F00000, 0x35F80000,
                0x36000000, 0x36040000, 0x36080000, 0x360C0000, 0x36100000, 0x36140000, 0x36180000, 0x361C0000, 0x36200000, 0x36240000, 0x36280000, 0x362C0000, 0x36300000, 0x36340000, 0x36380000, 0x363C0000,
                0x36400000, 0x36440000, 0x36480000, 0x364C0000, 0x36500000, 0x36540000, 0x36580000, 0x365C0000, 0x36600000, 0x36640000, 0x36680000, 0x366C0000, 0x36700000, 0x36740000, 0x36780000, 0x367C0000,
                0x36800000, 0x36820000, 0x36840000, 0x36860000, 0x36880000, 0x368A0000, 0x368C0000, 0x368E0000, 0x36900000, 0x36920000, 0x36940000, 0x36960000, 0x36980000, 0x369A0000, 0x369C0000, 0x369E0000,
                0x36A00000, 0x36A20000, 0x36A40000, 0x36A60000, 0x36A80000, 0x36AA0000, 0x36AC0000, 0x36AE0000, 0x36B00000, 0x36B20000, 0x36B40000, 0x36B60000, 0x36B80000, 0x36BA0000, 0x36BC0000, 0x36BE0000,
                0x36C00000, 0x36C20000, 0x36C40000, 0x36C60000, 0x36C80000, 0x36CA0000, 0x36CC0000, 0x36CE0000, 0x36D00000, 0x36D20000, 0x36D40000, 0x36D60000, 0x36D80000, 0x36DA0000, 0x36DC0000, 0x36DE0000,
                0x36E00000, 0x36E20000, 0x36E40000, 0x36E60000, 0x36E80000, 0x36EA0000, 0x36EC0000, 0x36EE0000, 0x36F00000, 0x36F20000, 0x36F40000, 0x36F60000, 0x36F80000, 0x36FA0000, 0x36FC0000, 0x36FE0000,
                0x37000000, 0x37010000, 0x37020000, 0x37030000, 0x37040000, 0x37050000, 0x37060000, 0x37070000, 0x37080000, 0x37090000, 0x370A0000, 0x370B0000, 0x370C0000, 0x370D0000, 0x370E0000, 0x370F0000,
                0x37100000, 0x37110000, 0x37120000, 0x37130000, 0x37140000, 0x37150000, 0x37160000, 0x37170000, 0x37180000, 0x37190000, 0x371A0000, 0x371B0000, 0x371C0000, 0x371D0000, 0x371E0000, 0x371F0000,
                0x37200000, 0x37210000, 0x37220000, 0x37230000, 0x37240000, 0x37250000, 0x37260000, 0x37270000, 0x37280000, 0x37290000, 0x372A0000, 0x372B0000, 0x372C0000, 0x372D0000, 0x372E0000, 0x372F0000,
                0x37300000, 0x37310000, 0x37320000, 0x37330000, 0x37340000, 0x37350000, 0x37360000, 0x37370000, 0x37380000, 0x37390000, 0x373A0000, 0x373B0000, 0x373C0000, 0x373D0000, 0x373E0000, 0x373F0000,
                0x37400000, 0x37410000, 0x37420000, 0x37430000, 0x37440000, 0x37450000, 0x37460000, 0x37470000, 0x37480000, 0x37490000, 0x374A0000, 0x374B0000, 0x374C0000, 0x374D0000, 0x374E0000,
0x374F0000,
           0x37500000, 0x37510000, 0x37520000, 0x37530000, 0x37540000, 0x37550000, 0x37560000,
0x37570000, 0x37580000, 0x37590000, 0x375A0000, 0x375B0000, 0x375C0000, 0x375D0000, 0x375E0000,
0x375F0000,
           0x37600000, 0x37610000, 0x37620000, 0x37630000, 0x37640000, 0x37650000, 0x37660000,
0x37670000, 0x37680000, 0x37690000, 0x376A0000, 0x376B0000, 0x376C0000, 0x376D0000, 0x376E0000,
0x376F0000,
           0x37700000, 0x37710000, 0x37720000, 0x37730000, 0x37740000, 0x37750000, 0x37760000,
0x37770000, 0x37780000, 0x37790000, 0x377A0000, 0x377B0000, 0x377C0000, 0x377D0000, 0x377E0000,
0x377F0000,
           0x37800000, 0x37808000, 0x37810000, 0x37818000, 0x37820000, 0x37828000, 0x37830000,
0x37838000, 0x37840000, 0x37848000, 0x37850000, 0x37858000, 0x37860000, 0x37868000, 0x37870000,
0x37878000,
           0x37880000, 0x37888000, 0x37890000, 0x37898000, 0x378A0000, 0x378A8000, 0x378B0000,
0x378B8000, 0x378C0000, 0x378C8000, 0x378D0000, 0x378D8000, 0x378E0000, 0x378E8000, 0x378F0000,
0x378F8000,
           0x37900000, 0x37908000, 0x37910000, 0x37918000, 0x37920000, 0x37928000, 0x37930000,
0x37938000, 0x37940000, 0x37948000, 0x37950000, 0x37958000, 0x37960000, 0x37968000, 0x37970000,
0x37978000,
           0x37980000, 0x37988000, 0x37990000, 0x37998000, 0x379A0000, 0x379A8000, 0x379B0000,
0x379B8000, 0x379C0000, 0x379C8000, 0x379D0000, 0x379D8000, 0x379E0000, 0x379E8000, 0x379F0000,
0x379F8000,
           0x37A00000, 0x37A08000, 0x37A10000, 0x37A18000, 0x37A20000, 0x37A28000, 0x37A30000,
0x37A38000, 0x37A40000, 0x37A48000, 0x37A50000, 0x37A58000, 0x37A60000, 0x37A68000, 0x37A70000,
0x37A78000,
           0x37A80000, 0x37A88000, 0x37A90000, 0x37A98000, 0x37AA0000, 0x37AA8000, 0x37AB0000,
0x37AB8000, 0x37AC0000, 0x37AC8000, 0x37AD0000, 0x37AD8000, 0x37AE0000, 0x37AE8000, 0x37AF0000,
0x37AF8000,
           0x37B00000, 0x37B08000, 0x37B10000, 0x37B18000, 0x37B20000, 0x37B28000, 0x37B30000,
0x37B38000, 0x37B40000, 0x37B48000, 0x37B50000, 0x37B58000, 0x37B60000, 0x37B68000, 0x37B70000,
0x37B78000,
           0x37B80000, 0x37B88000, 0x37B90000, 0x37B98000, 0x37BA0000, 0x37BA8000, 0x37BB0000,
0x37BB8000, 0x37BC0000, 0x37BC8000, 0x37BD0000, 0x37BD8000, 0x37BE0000, 0x37BE8000, 0x37BF0000,
0x37BF8000,
           0x37C00000, 0x37C08000, 0x37C10000, 0x37C18000, 0x37C20000, 0x37C28000, 0x37C30000,
0x37C38000, 0x37C40000, 0x37C48000, 0x37C50000, 0x37C58000, 0x37C60000, 0x37C68000, 0x37C70000,
0x37C78000,
           0x37C80000, 0x37C88000, 0x37C90000, 0x37C98000, 0x37CA0000, 0x37CA8000, 0x37CB0000,
0x37CB8000, 0x37CC0000, 0x37CC8000, 0x37CD0000, 0x37CD8000, 0x37CE0000, 0x37CE8000, 0x37CF0000,
0x37CF8000,
           0x37D00000, 0x37D08000, 0x37D10000, 0x37D18000, 0x37D20000, 0x37D28000, 0x37D30000,
0x37D38000, 0x37D40000, 0x37D48000, 0x37D50000, 0x37D58000, 0x37D60000, 0x37D68000, 0x37D70000,
0x37D78000,
           0x37D80000, 0x37D88000, 0x37D90000, 0x37D98000, 0x37DA0000, 0x37DA8000, 0x37DB0000,
0x37DB8000, 0x37DC0000, 0x37DC8000, 0x37DD0000, 0x37DD8000, 0x37DE0000, 0x37DE8000, 0x37DF0000,
0x37DF8000,
           0x37E00000, 0x37E08000, 0x37E10000, 0x37E18000, 0x37E20000, 0x37E28000, 0x37E30000,
0x37E38000, 0x37E40000, 0x37E48000, 0x37E50000, 0x37E58000, 0x37E60000, 0x37E68000, 0x37E70000,
0x37E78000,
           0x37E80000, 0x37E88000, 0x37E90000, 0x37E98000, 0x37EA0000, 0x37EA8000, 0x37EB0000,
0x37EB8000, 0x37EC0000, 0x37EC8000, 0x37ED0000, 0x37ED8000, 0x37EE0000, 0x37EE8000, 0x37EF0000,
0x37EF8000,
           0x37F00000, 0x37F08000, 0x37F10000, 0x37F18000, 0x37F20000, 0x37F28000, 0x37F30000,
0x37F38000, 0x37F40000, 0x37F48000, 0x37F50000, 0x37F58000, 0x37F60000, 0x37F68000, 0x37F70000,
0x37F78000,
           0x37F80000, 0x37F88000, 0x37F90000, 0x37F98000, 0x37FA0000, 0x37FA8000, 0x37FB0000,
0x37FB8000, 0x37FC0000, 0x37FC8000, 0x37FD0000, 0x37FD8000, 0x37FE0000, 0x37FE8000, 0x37FF0000,
0x37FF8000,
           0x38000000, 0x38004000, 0x38008000, 0x3800C000, 0x38010000, 0x38014000, 0x38018000,
0x3801C000, 0x38020000, 0x38024000, 0x38028000, 0x3802C000, 0x38030000, 0x38034000, 0x38038000,
0x3803C000,
           0x38040000, 0x38044000, 0x38048000, 0x3804C000, 0x38050000, 0x38054000, 0x38058000,
0x3805C000, 0x38060000, 0x38064000, 0x38068000, 0x3806C000, 0x38070000, 0x38074000, 0x38078000,
0x3807C000,
           0x38080000, 0x38084000, 0x38088000, 0x3808C000, 0x38090000, 0x38094000, 0x38098000,
0x3809C000, 0x380A0000, 0x380A4000, 0x380A8000, 0x380AC000, 0x380B0000, 0x380B4000, 0x380B8000,
0x380BC000,
           0x380C0000, 0x380C4000, 0x380C8000, 0x380CC000, 0x380D0000, 0x380D4000, 0x380D8000,
0x380DC000, 0x380E0000, 0x380E4000, 0x380E8000, 0x380EC000, 0x380F0000, 0x380F4000, 0x380F8000,
0x380FC000,
           0x38100000, 0x38104000, 0x38108000, 0x3810C000, 0x38110000, 0x38114000, 0x38118000,
0x3811C000, 0x38120000, 0x38124000, 0x38128000, 0x3812C000, 0x38130000, 0x38134000, 0x38138000,
0x3813C000,
           0x38140000, 0x38144000, 0x38148000, 0x3814C000, 0x38150000, 0x38154000, 0x38158000,
0x3815C000, 0x38160000, 0x38164000, 0x38168000, 0x3816C000, 0x38170000, 0x38174000, 0x38178000,
0x3817C000,
           0x38180000, 0x38184000, 0x38188000, 0x3818C000, 0x38190000, 0x38194000, 0x38198000,
0x3819C000, 0x381A0000, 0x381A4000, 0x381A8000, 0x381AC000, 0x381B0000, 0x381B4000, 0x381B8000,
0x381BC000,
           0x381C0000, 0x381C4000, 0x381C8000, 0x381CC000, 0x381D0000, 0x381D4000, 0x381D8000,
0x381DC000, 0x381E0000, 0x381E4000, 0x381E8000, 0x381EC000, 0x381F0000, 0x381F4000, 0x381F8000,
0x381FC000,
           0x38200000, 0x38204000, 0x38208000, 0x3820C000, 0x38210000, 0x38214000, 0x38218000,
0x3821C000, 0x38220000, 0x38224000, 0x38228000, 0x3822C000, 0x38230000, 0x38234000, 0x38238000,
0x3823C000,
           0x38240000, 0x38244000, 0x38248000, 0x3824C000, 0x38250000, 0x38254000, 0x38258000,
0x3825C000, 0x38260000, 0x38264000, 0x38268000, 0x3826C000, 0x38270000, 0x38274000, 0x38278000,
0x3827C000,
                0x38280000, 0x38284000, 0x38288000, 0x3828C000, 0x38290000, 0x38294000, 0x38298000,
0x3829C000, 0x382A0000, 0x382A4000, 0x382A8000, 0x382AC000, 0x382B0000, 0x382B4000, 0x382B8000,
0x382BC000,
                0x382C0000, 0x382C4000, 0x382C8000, 0x382CC000, 0x382D0000, 0x382D4000, 0x382D8000,
0x382DC000, 0x382E0000, 0x382E4000, 0x382E8000, 0x382EC000, 0x382F0000, 0x382F4000, 0x382F8000,
0x382FC000,
                0x38300000, 0x38304000, 0x38308000, 0x3830C000, 0x38310000, 0x38314000, 0x38318000,
0x3831C000, 0x38320000, 0x38324000, 0x38328000, 0x3832C000, 0x38330000, 0x38334000, 0x38338000,
0x3833C000,
                0x38340000, 0x38344000, 0x38348000, 0x3834C000, 0x38350000, 0x38354000, 0x38358000,
0x3835C000, 0x38360000, 0x38364000, 0x38368000, 0x3836C000, 0x38370000, 0x38374000, 0x38378000,
0x3837C000,
                0x38380000, 0x38384000, 0x38388000, 0x3838C000, 0x38390000, 0x38394000, 0x38398000,
0x3839C000, 0x383A0000, 0x383A4000, 0x383A8000, 0x383AC000, 0x383B0000, 0x383B4000, 0x383B8000,
0x383BC000,
                0x383C0000, 0x383C4000, 0x383C8000, 0x383CC000, 0x383D0000, 0x383D4000, 0x383D8000,
0x383DC000, 0x383E0000, 0x383E4000, 0x383E8000, 0x383EC000, 0x383F0000, 0x383F4000, 0x383F8000,
0x383FC000,
                0x38400000, 0x38404000, 0x38408000, 0x3840C000, 0x38410000, 0x38414000, 0x38418000,
0x3841C000, 0x38420000, 0x38424000, 0x38428000, 0x3842C000, 0x38430000, 0x38434000, 0x38438000,
0x3843C000,
                0x38440000, 0x38444000, 0x38448000, 0x3844C000, 0x38450000, 0x38454000, 0x38458000,
0x3845C000, 0x38460000, 0x38464000, 0x38468000, 0x3846C000, 0x38470000, 0x38474000, 0x38478000,
0x3847C000,
                0x38480000, 0x38484000, 0x38488000, 0x3848C000, 0x38490000, 0x38494000, 0x38498000,
0x3849C000, 0x384A0000, 0x384A4000, 0x384A8000, 0x384AC000, 0x384B0000, 0x384B4000, 0x384B8000,
0x384BC000,
                0x384C0000, 0x384C4000, 0x384C8000, 0x384CC000, 0x384D0000, 0x384D4000, 0x384D8000,
0x384DC000, 0x384E0000, 0x384E4000, 0x384E8000, 0x384EC000, 0x384F0000, 0x384F4000, 0x384F8000,
0x384FC000,
                0x38500000, 0x38504000, 0x38508000, 0x3850C000, 0x38510000, 0x38514000, 0x38518000,
0x3851C000, 0x38520000, 0x38524000, 0x38528000, 0x3852C000, 0x38530000, 0x38534000, 0x38538000,
0x3853C000,
                0x38540000, 0x38544000, 0x38548000, 0x3854C000, 0x38550000, 0x38554000, 0x38558000,
0x3855C000, 0x38560000, 0x38564000, 0x38568000, 0x3856C000, 0x38570000, 0x38574000, 0x38578000,
0x3857C000,
                0x38580000, 0x38584000, 0x38588000, 0x3858C000, 0x38590000, 0x38594000, 0x38598000,
0x3859C000, 0x385A0000, 0x385A4000, 0x385A8000, 0x385AC000, 0x385B0000, 0x385B4000, 0x385B8000,
0x385BC000,
                0x385C0000, 0x385C4000, 0x385C8000, 0x385CC000, 0x385D0000, 0x385D4000, 0x385D8000,
0x385DC000, 0x385E0000, 0x385E4000, 0x385E8000, 0x385EC000, 0x385F0000, 0x385F4000, 0x385F8000,
0x385FC000,
                0x38600000, 0x38604000, 0x38608000, 0x3860C000, 0x38610000, 0x38614000, 0x38618000,
0x3861C000, 0x38620000, 0x38624000, 0x38628000, 0x3862C000, 0x38630000, 0x38634000, 0x38638000,
0x3863C000,
                0x38640000, 0x38644000, 0x38648000, 0x3864C000, 0x38650000, 0x38654000, 0x38658000,
0x3865C000, 0x38660000, 0x38664000, 0x38668000, 0x3866C000, 0x38670000, 0x38674000, 0x38678000,
0x3867C000,
                0x38680000, 0x38684000, 0x38688000, 0x3868C000, 0x38690000, 0x38694000, 0x38698000,
0x3869C000, 0x386A0000, 0x386A4000, 0x386A8000, 0x386AC000, 0x386B0000, 0x386B4000, 0x386B8000,
0x386BC000,
                0x386C0000, 0x386C4000, 0x386C8000, 0x386CC000, 0x386D0000, 0x386D4000, 0x386D8000,
0x386DC000, 0x386E0000, 0x386E4000, 0x386E8000, 0x386EC000, 0x386F0000, 0x386F4000, 0x386F8000,
0x386FC000,
                0x38700000, 0x38704000, 0x38708000, 0x3870C000, 0x38710000, 0x38714000, 0x38718000,
0x3871C000, 0x38720000, 0x38724000, 0x38728000, 0x3872C000, 0x38730000, 0x38734000, 0x38738000,
0x3873C000,
                0x38740000, 0x38744000, 0x38748000, 0x3874C000, 0x38750000, 0x38754000, 0x38758000,
0x3875C000, 0x38760000, 0x38764000, 0x38768000, 0x3876C000, 0x38770000, 0x38774000, 0x38778000,
0x3877C000,
                0x38780000, 0x38784000, 0x38788000, 0x3878C000, 0x38790000, 0x38794000, 0x38798000,
0x3879C000, 0x387A0000, 0x387A4000, 0x387A8000, 0x387AC000, 0x387B0000, 0x387B4000, 0x387B8000,
0x387BC000,
                0x387C0000, 0x387C4000, 0x387C8000, 0x387CC000, 0x387D0000, 0x387D4000, 0x387D8000,
0x387DC000, 0x387E0000, 0x387E4000, 0x387E8000, 0x387EC000, 0x387F0000, 0x387F4000, 0x387F8000,
0x387FC000,
                0x38000000, 0x38002000, 0x38004000, 0x38006000, 0x38008000, 0x3800A000, 0x3800C000,
0x3800E000, 0x38010000, 0x38012000, 0x38014000, 0x38016000, 0x38018000, 0x3801A000, 0x3801C000,
0x3801E000,
                0x38020000, 0x38022000, 0x38024000, 0x38026000, 0x38028000, 0x3802A000, 0x3802C000,
0x3802E000, 0x38030000, 0x38032000, 0x38034000, 0x38036000, 0x38038000, 0x3803A000, 0x3803C000,
0x3803E000,
                0x38040000, 0x38042000, 0x38044000, 0x38046000, 0x38048000, 0x3804A000, 0x3804C000,
0x3804E000, 0x38050000, 0x38052000, 0x38054000, 0x38056000, 0x38058000, 0x3805A000, 0x3805C000,
0x3805E000,
                0x38060000, 0x38062000, 0x38064000, 0x38066000, 0x38068000, 0x3806A000, 0x3806C000,
0x3806E000, 0x38070000, 0x38072000, 0x38074000, 0x38076000, 0x38078000, 0x3807A000, 0x3807C000,
0x3807E000,
                0x38080000, 0x38082000, 0x38084000, 0x38086000, 0x38088000, 0x3808A000, 0x3808C000,
0x3808E000, 0x38090000, 0x38092000, 0x38094000, 0x38096000, 0x38098000, 0x3809A000, 0x3809C000,
0x3809E000,
                0x380A0000, 0x380A2000, 0x380A4000, 0x380A6000, 0x380A8000, 0x380AA000, 0x380AC000,
0x380AE000, 0x380B0000, 0x380B2000, 0x380B4000, 0x380B6000, 0x380B8000, 0x380BA000, 0x380BC000,
0x380BE000,
                0x380C0000, 0x380C2000, 0x380C4000, 0x380C6000, 0x380C8000, 0x380CA000, 0x380CC000,
0x380CE000, 0x380D0000, 0x380D2000, 0x380D4000, 0x380D6000, 0x380D8000, 0x380DA000, 0x380DC000,
0x380DE000,
                  0x380E0000, 0x380E2000, 0x380E4000, 0x380E6000, 0x380E8000, 0x380EA000, 0x380EC000,
0x380EE000, 0x380F0000, 0x380F2000, 0x380F4000, 0x380F6000, 0x380F8000, 0x380FA000, 0x380FC000,
0x380FE000,
                  0x38100000, 0x38102000, 0x38104000, 0x38106000, 0x38108000, 0x3810A000, 0x3810C000,
0x3810E000, 0x38110000, 0x38112000, 0x38114000, 0x38116000, 0x38118000, 0x3811A000, 0x3811C000,
0x3811E000,
                  0x38120000, 0x38122000, 0x38124000, 0x38126000, 0x38128000, 0x3812A000, 0x3812C000,
0x3812E000, 0x38130000, 0x38132000, 0x38134000, 0x38136000, 0x38138000, 0x3813A000, 0x3813C000,
0x3813E000,
                  0x38140000, 0x38142000, 0x38144000, 0x38146000, 0x38148000, 0x3814A000, 0x3814C000,
0x3814E000, 0x38150000, 0x38152000, 0x38154000, 0x38156000, 0x38158000, 0x3815A000, 0x3815C000,
0x3815E000,
                  0x38160000, 0x38162000, 0x38164000, 0x38166000, 0x38168000, 0x3816A000, 0x3816C000,
0x3816E000, 0x38170000, 0x38172000, 0x38174000, 0x38176000, 0x38178000, 0x3817A000, 0x3817C000,
0x3817E000,
                  0x38180000, 0x38182000, 0x38184000, 0x38186000, 0x38188000, 0x3818A000, 0x3818C000,
0x3818E000, 0x38190000, 0x38192000, 0x38194000, 0x38196000, 0x38198000, 0x3819A000, 0x3819C000,
0x3819E000,
                  0x381A0000, 0x381A2000, 0x381A4000, 0x381A6000, 0x381A8000, 0x381AA000, 0x381AC000,
0x381AE000, 0x381B0000, 0x381B2000, 0x381B4000, 0x381B6000, 0x381B8000, 0x381BA000, 0x381BC000,
0x381BE000,
                  0x381C0000, 0x381C2000, 0x381C4000, 0x381C6000, 0x381C8000, 0x381CA000, 0x381CC000,
0x381CE000, 0x381D0000, 0x381D2000, 0x381D4000, 0x381D6000, 0x381D8000, 0x381DA000, 0x381DC000,
0x381DE000,
                  0x381E0000, 0x381E2000, 0x381E4000, 0x381E6000, 0x381E8000, 0x381EA000, 0x381EC000,
0x381EE000, 0x381F0000, 0x381F2000, 0x381F4000, 0x381F6000, 0x381F8000, 0x381FA000, 0x381FC000,
0x381FE000,
                  0x38200000, 0x38202000, 0x38204000, 0x38206000, 0x38208000, 0x3820A000, 0x3820C000,
0x3820E000, 0x38210000, 0x38212000, 0x38214000, 0x38216000, 0x38218000, 0x3821A000, 0x3821C000,
0x3821E000,
                  0x38220000, 0x38222000, 0x38224000, 0x38226000, 0x38228000, 0x3822A000, 0x3822C000,
0x3822E000, 0x38230000, 0x38232000, 0x38234000, 0x38236000, 0x38238000, 0x3823A000, 0x3823C000,
0x3823E000,
                  0x38240000, 0x38242000, 0x38244000, 0x38246000, 0x38248000, 0x3824A000, 0x3824C000,
0x3824E000, 0x38250000, 0x38252000, 0x38254000, 0x38256000, 0x38258000, 0x3825A000, 0x3825C000,
0x3825E000,
                  0x38260000, 0x38262000, 0x38264000, 0x38266000, 0x38268000, 0x3826A000, 0x3826C000,
0x3826E000, 0x38270000, 0x38272000, 0x38274000, 0x38276000, 0x38278000, 0x3827A000, 0x3827C000,
0x3827E000,
                  0x38280000, 0x38282000, 0x38284000, 0x38286000, 0x38288000, 0x3828A000, 0x3828C000,
0x3828E000, 0x38290000, 0x38292000, 0x38294000, 0x38296000, 0x38298000, 0x3829A000, 0x3829C000,
0x3829E000,
                  0x382A0000, 0x382A2000, 0x382A4000, 0x382A6000, 0x382A8000, 0x382AA000, 0x382AC000,
0x382AE000, 0x382B0000, 0x382B2000, 0x382B4000, 0x382B6000, 0x382B8000, 0x382BA000, 0x382BC000,
0x382BE000,
                  0x382C0000, 0x382C2000, 0x382C4000, 0x382C6000, 0x382C8000, 0x382CA000, 0x382CC000,
0x382CE000, 0x382D0000, 0x382D2000, 0x382D4000, 0x382D6000, 0x382D8000, 0x382DA000, 0x382DC000,
0x382DE000,
                  0x382E0000, 0x382E2000, 0x382E4000, 0x382E6000, 0x382E8000, 0x382EA000, 0x382EC000,
0x382EE000, 0x382F0000, 0x382F2000, 0x382F4000, 0x382F6000, 0x382F8000, 0x382FA000, 0x382FC000,
0x382FE000,
                  0x38300000, 0x38302000, 0x38304000, 0x38306000, 0x38308000, 0x3830A000, 0x3830C000,
0x3830E000, 0x38310000, 0x38312000, 0x38314000, 0x38316000, 0x38318000, 0x3831A000, 0x3831C000,
0x3831E000,
                  0x38320000, 0x38322000, 0x38324000, 0x38326000, 0x38328000, 0x3832A000, 0x3832C000,
0x3832E000, 0x38330000, 0x38332000, 0x38334000, 0x38336000, 0x38338000, 0x3833A000, 0x3833C000,
0x3833E000,
                  0x38340000, 0x38342000, 0x38344000, 0x38346000, 0x38348000, 0x3834A000, 0x3834C000,
0x3834E000, 0x38350000, 0x38352000, 0x38354000, 0x38356000, 0x38358000, 0x3835A000, 0x3835C000,
0x3835E000,
                  0x38360000, 0x38362000, 0x38364000, 0x38366000, 0x38368000, 0x3836A000, 0x3836C000,
0x3836E000, 0x38370000, 0x38372000, 0x38374000, 0x38376000, 0x38378000, 0x3837A000, 0x3837C000,
0x3837E000,
                  0x38380000, 0x38382000, 0x38384000, 0x38386000, 0x38388000, 0x3838A000, 0x3838C000,
0x3838E000, 0x38390000, 0x38392000, 0x38394000, 0x38396000, 0x38398000, 0x3839A000, 0x3839C000,
0x3839E000,
                  0x383A0000, 0x383A2000, 0x383A4000, 0x383A6000, 0x383A8000, 0x383AA000, 0x383AC000,
0x383AE000, 0x383B0000, 0x383B2000, 0x383B4000, 0x383B6000, 0x383B8000, 0x383BA000, 0x383BC000,
0x383BE000,
                  0x383C0000, 0x383C2000, 0x383C4000, 0x383C6000, 0x383C8000, 0x383CA000, 0x383CC000,
0x383CE000, 0x383D0000, 0x383D2000, 0x383D4000, 0x383D6000, 0x383D8000, 0x383DA000, 0x383DC000,
0x383DE000,
                  0x383E0000, 0x383E2000, 0x383E4000, 0x383E6000, 0x383E8000, 0x383EA000, 0x383EC000,
0x383EE000, 0x383F0000, 0x383F2000, 0x383F4000, 0x383F6000, 0x383F8000, 0x383FA000, 0x383FC000,
0x383FE000,
                  0x38400000, 0x38402000, 0x38404000, 0x38406000, 0x38408000, 0x3840A000, 0x3840C000,
0x3840E000, 0x38410000, 0x38412000, 0x38414000, 0x38416000, 0x38418000, 0x3841A000, 0x3841C000,
0x3841E000,
                  0x38420000, 0x38422000, 0x38424000, 0x38426000, 0x38428000, 0x3842A000, 0x3842C000,
0x3842E000, 0x38430000, 0x38432000, 0x38434000, 0x38436000, 0x38438000, 0x3843A000, 0x3843C000,
0x3843E000,
                  0x38440000, 0x38442000, 0x38444000, 0x38446000, 0x38448000, 0x3844A000, 0x3844C000,
0x3844E000, 0x38450000, 0x38452000, 0x38454000, 0x38456000, 0x38458000, 0x3845A000, 0x3845C000,
0x3845E000,
                  0x38460000, 0x38462000, 0x38464000, 0x38466000, 0x38468000, 0x3846A000, 0x3846C000,
0x3846E000, 0x38470000, 0x38472000, 0x38474000, 0x38476000, 0x38478000, 0x3847A000, 0x3847C000,
0x3847E000,
        0x38480000, 0x38482000, 0x38484000, 0x38486000, 0x38488000, 0x3848A000, 0x3848C000,
0x3848E000, 0x38490000, 0x38492000, 0x38494000, 0x38496000, 0x38498000, 0x3849A000, 0x3849C000,
0x3849E000,
        0x384A0000, 0x384A2000, 0x384A4000, 0x384A6000, 0x384A8000, 0x384AA000, 0x384AC000,
0x384AE000, 0x384B0000, 0x384B2000, 0x384B4000, 0x384B6000, 0x384B8000, 0x384BA000, 0x384BC000,
0x384BE000,
        0x384C0000, 0x384C2000, 0x384C4000, 0x384C6000, 0x384C8000, 0x384CA000, 0x384CC000,
0x384CE000, 0x384D0000, 0x384D2000, 0x384D4000, 0x384D6000, 0x384D8000, 0x384DA000, 0x384DC000,
0x384DE000,
        0x384E0000, 0x384E2000, 0x384E4000, 0x384E6000, 0x384E8000, 0x384EA000, 0x384EC000,
0x384EE000, 0x384F0000, 0x384F2000, 0x384F4000, 0x384F6000, 0x384F8000, 0x384FA000, 0x384FC000,
0x384FE000,
        0x38500000, 0x38502000, 0x38504000, 0x38506000, 0x38508000, 0x3850A000, 0x3850C000,
0x3850E000, 0x38510000, 0x38512000, 0x38514000, 0x38516000, 0x38518000, 0x3851A000, 0x3851C000,
0x3851E000,
        0x38520000, 0x38522000, 0x38524000, 0x38526000, 0x38528000, 0x3852A000, 0x3852C000,
0x3852E000, 0x38530000, 0x38532000, 0x38534000, 0x38536000, 0x38538000, 0x3853A000, 0x3853C000,
0x3853E000,
        0x38540000, 0x38542000, 0x38544000, 0x38546000, 0x38548000, 0x3854A000, 0x3854C000,
0x3854E000, 0x38550000, 0x38552000, 0x38554000, 0x38556000, 0x38558000, 0x3855A000, 0x3855C000,
0x3855E000,
        0x38560000, 0x38562000, 0x38564000, 0x38566000, 0x38568000, 0x3856A000, 0x3856C000,
0x3856E000, 0x38570000, 0x38572000, 0x38574000, 0x38576000, 0x38578000, 0x3857A000, 0x3857C000,
0x3857E000,
        0x38580000, 0x38582000, 0x38584000, 0x38586000, 0x38588000, 0x3858A000, 0x3858C000,
0x3858E000, 0x38590000, 0x38592000, 0x38594000, 0x38596000, 0x38598000, 0x3859A000, 0x3859C000,
0x3859E000,
        0x385A0000, 0x385A2000, 0x385A4000, 0x385A6000, 0x385A8000, 0x385AA000, 0x385AC000,
0x385AE000, 0x385B0000, 0x385B2000, 0x385B4000, 0x385B6000, 0x385B8000, 0x385BA000, 0x385BC000,
0x385BE000,
        0x385C0000, 0x385C2000, 0x385C4000, 0x385C6000, 0x385C8000, 0x385CA000, 0x385CC000,
0x385CE000, 0x385D0000, 0x385D2000, 0x385D4000, 0x385D6000, 0x385D8000, 0x385DA000, 0x385DC000,
0x385DE000,
        0x385E0000, 0x385E2000, 0x385E4000, 0x385E6000, 0x385E8000, 0x385EA000, 0x385EC000,
0x385EE000, 0x385F0000, 0x385F2000, 0x385F4000, 0x385F6000, 0x385F8000, 0x385FA000, 0x385FC000,
0x385FE000,
        0x38600000, 0x38602000, 0x38604000, 0x38606000, 0x38608000, 0x3860A000, 0x3860C000,
0x3860E000, 0x38610000, 0x38612000, 0x38614000, 0x38616000, 0x38618000, 0x3861A000, 0x3861C000,
0x3861E000,
        0x38620000, 0x38622000, 0x38624000, 0x38626000, 0x38628000, 0x3862A000, 0x3862C000,
0x3862E000, 0x38630000, 0x38632000, 0x38634000, 0x38636000, 0x38638000, 0x3863A000, 0x3863C000,
0x3863E000,
        0x38640000, 0x38642000, 0x38644000, 0x38646000, 0x38648000, 0x3864A000, 0x3864C000,
0x3864E000, 0x38650000, 0x38652000, 0x38654000, 0x38656000, 0x38658000, 0x3865A000, 0x3865C000,
0x3865E000,
        0x38660000, 0x38662000, 0x38664000, 0x38666000, 0x38668000, 0x3866A000, 0x3866C000,
0x3866E000, 0x38670000, 0x38672000, 0x38674000, 0x38676000, 0x38678000, 0x3867A000, 0x3867C000,
0x3867E000,
        0x38680000, 0x38682000, 0x38684000, 0x38686000, 0x38688000, 0x3868A000, 0x3868C000,
0x3868E000, 0x38690000, 0x38692000, 0x38694000, 0x38696000, 0x38698000, 0x3869A000, 0x3869C000,
0x3869E000,
        0x386A0000, 0x386A2000, 0x386A4000, 0x386A6000, 0x386A8000, 0x386AA000, 0x386AC000,
0x386AE000, 0x386B0000, 0x386B2000, 0x386B4000, 0x386B6000, 0x386B8000, 0x386BA000, 0x386BC000,
0x386BE000,
        0x386C0000, 0x386C2000, 0x386C4000, 0x386C6000, 0x386C8000, 0x386CA000, 0x386CC000,
0x386CE000, 0x386D0000, 0x386D2000, 0x386D4000, 0x386D6000, 0x386D8000, 0x386DA000, 0x386DC000,
0x386DE000,
        0x386E0000, 0x386E2000, 0x386E4000, 0x386E6000, 0x386E8000, 0x386EA000, 0x386EC000,
0x386EE000, 0x386F0000, 0x386F2000, 0x386F4000, 0x386F6000, 0x386F8000, 0x386FA000, 0x386FC000,
0x386FE000,
        0x38700000, 0x38702000, 0x38704000, 0x38706000, 0x38708000, 0x3870A000, 0x3870C000,
0x3870E000, 0x38710000, 0x38712000, 0x38714000, 0x38716000, 0x38718000, 0x3871A000, 0x3871C000,
0x3871E000,
        0x38720000, 0x38722000, 0x38724000, 0x38726000, 0x38728000, 0x3872A000, 0x3872C000,
0x3872E000, 0x38730000, 0x38732000, 0x38734000, 0x38736000, 0x38738000, 0x3873A000, 0x3873C000,
0x3873E000,
        0x38740000, 0x38742000, 0x38744000, 0x38746000, 0x38748000, 0x3874A000, 0x3874C000,
0x3874E000, 0x38750000, 0x38752000, 0x38754000, 0x38756000, 0x38758000, 0x3875A000, 0x3875C000,
0x3875E000,
        0x38760000, 0x38762000, 0x38764000, 0x38766000, 0x38768000, 0x3876A000, 0x3876C000,
0x3876E000, 0x38770000, 0x38772000, 0x38774000, 0x38776000, 0x38778000, 0x3877A000, 0x3877C000,
0x3877E000,
        0x38780000, 0x38782000, 0x38784000, 0x38786000, 0x38788000, 0x3878A000, 0x3878C000,
0x3878E000, 0x38790000, 0x38792000, 0x38794000, 0x38796000, 0x38798000, 0x3879A000, 0x3879C000,
0x3879E000,
        0x387A0000, 0x387A2000, 0x387A4000, 0x387A6000, 0x387A8000, 0x387AA000, 0x387AC000,
0x387AE000, 0x387B0000, 0x387B2000, 0x387B4000, 0x387B6000, 0x387B8000, 0x387BA000, 0x387BC000,
0x387BE000,
        0x387C0000, 0x387C2000, 0x387C4000, 0x387C6000, 0x387C8000, 0x387CA000, 0x387CC000,
0x387CE000, 0x387D0000, 0x387D2000, 0x387D4000, 0x387D6000, 0x387D8000, 0x387DA000, 0x387DC000,
0x387DE000,
        0x387E0000, 0x387E2000, 0x387E4000, 0x387E6000, 0x387E8000, 0x387EA000, 0x387EC000,
0x387EE000, 0x387F0000, 0x387F2000, 0x387F4000, 0x387F6000, 0x387F8000, 0x387FA000, 0x387FC000,
0x387FE000 };
    static const bits<float>::type exponent_table[64] = {
        0x00000000, 0x00800000, 0x01000000, 0x01800000, 0x02000000, 0x02800000, 0x03000000,
0x03800000, 0x04000000, 0x04800000, 0x05000000, 0x05800000, 0x06000000, 0x06800000, 0x07000000, 0x07800000,
        0x08000000, 0x08800000, 0x09000000, 0x09800000, 0x0A000000, 0x0A800000, 0x0B000000, 0x0B800000, 0x0C000000, 0x0C800000, 0x0D000000, 0x0D800000, 0x0E000000, 0x0E800000, 0x0F000000, 0x47800000,
        0x80000000, 0x80800000, 0x81000000, 0x81800000, 0x82000000, 0x82800000, 0x83000000, 0x83800000, 0x84000000, 0x84800000, 0x85000000, 0x85800000, 0x86000000, 0x86800000, 0x87000000, 0x87800000,
        0x88000000, 0x88800000, 0x89000000, 0x89800000, 0x8A000000, 0x8A800000, 0x8B000000, 0x8B800000, 0x8C000000, 0x8C800000, 0x8D000000, 0x8D800000, 0x8E000000, 0x8E800000, 0x8F000000, 0xC7800000 };
    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,
        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 };
    bits<float>::type fbits = mantissa_table[offset_table[value»10]+(value&0x3FF)] + exponent_table[value»10];
#endif
    float out;
    std::memcpy(&out, &fbits, sizeof(float));
    return out;
#endif
}

inline double half2float_impl(unsigned int value, double, true_type)
{
#if HALF_ENABLE_F16C_INTRINSICS
    return _mm_cvtsd_f64(_mm_cvtps_pd(_mm_cvtph_ps(_mm_cvtsi32_si128(value))));
#else
    uint32 hi = static_cast<uint32>(value&0x8000) « 16;
    unsigned int abs = value & 0x7FFF;
    if(abs)
    {
        hi |= 0x3F000000 « static_cast<unsigned>(abs>=0x7C00);
        for(; abs<0x400; abs«=1,hi-=0x100000) ;
        hi += static_cast<uint32>(abs) « 10;
    }
    bits<double>::type dbits = static_cast<bits<double>::type>(hi) « 32;
    double out;
    std::memcpy(&out, &dbits, sizeof(double));
    return out;
#endif
}

template<typename T> T half2float_impl(unsigned int value, T, ...)
{
    T out;
    unsigned int abs = value & 0x7FFF;
    if(abs > 0x7C00)
        out = (std::numeric_limits<T>::has_signaling_NaN && !(abs&0x200)) ? std::numeric_limits<T>::signaling_NaN() :
            std::numeric_limits<T>::has_quiet_NaN ? std::numeric_limits<T>::quiet_NaN() : T();
    else if(abs == 0x7C00)
        out = std::numeric_limits<T>::has_infinity ? std::numeric_limits<T>::infinity() : std::numeric_limits<T>::max();
    else if(abs > 0x3FF)
        out = std::ldexp(static_cast<T>((abs&0x3FF)|0x400), (abs»10)-25);
    else
        out = std::ldexp(static_cast<T>(abs), -24);
    return (value&0x8000) ? -out : out;
}

template<typename T> T half2float(unsigned int value)
{
    return half2float_impl(value, T(), bool_type<std::numeric_limits<T>::is_iec559&&sizeof(typename bits<T>::type)==sizeof(T)>());
}

template<std::float_round_style R,bool E,bool I,typename T> T half2int(unsigned int value)
{
    unsigned int abs = value & 0x7FFF;
    if(abs >= 0x7C00)
    {
        raise(FE_INVALID);
        return (value&0x8000) ? std::numeric_limits<T>::min() : std::numeric_limits<T>::max();
    }
    if(abs < 0x3800)
    {
        raise(FE_INEXACT, I);
        return  (R==std::round_toward_infinity) ? T(~(value»15)&(abs!=0)) :
                (R==std::round_toward_neg_infinity) ? -T(value>0x8000) :
                T();
    }
    int exp = 25 - (abs»10);
    unsigned int m = (value&0x3FF) | 0x400;
    int32 i = static_cast<int32>((exp<=0) ? (m«-exp) : ((m+(
        (R==std::round_to_nearest) ? ((1«(exp-1))-(~(m»exp)&E)) :
        (R==std::round_toward_infinity) ? (((1«exp)-1)&((value»15)-1)) :
        (R==std::round_toward_neg_infinity) ? (((1«exp)-1)&-(value»15)) : 0))»exp));
    if((!std::numeric_limits<T>::is_signed && (value&0x8000)) || (std::numeric_limits<T>::digits<16 &&
        ((value&0x8000) ? (-i<std::numeric_limits<T>::min()) : (i>std::numeric_limits<T>::max()))))
        raise(FE_INVALID);
    else if(I && exp > 0 && (m&((1«exp)-1)))
        raise(FE_INEXACT);
    return static_cast<T>((value&0x8000) ? -i : i);
}


template<std::float_round_style R> uint32 mulhi(uint32 x, uint32 y)
{
    uint32 xy = (x»16) * (y&0xFFFF), yx = (x&0xFFFF) * (y»16), c = (xy&0xFFFF) + (yx&0xFFFF) + (((x&0xFFFF)*(y&0xFFFF))»16);
    return (x»16)*(y»16) + (xy»16) + (yx»16) + (c»16) +
        ((R==std::round_to_nearest) ? ((c»15)&1) : (R==std::round_toward_infinity) ? ((c&0xFFFF)!=0) : 0);
}

inline uint32 multiply64(uint32 x, uint32 y)
{
#if HALF_ENABLE_CPP11_LONG_LONG
    return static_cast<uint32>((static_cast<unsigned long long>(x)*static_cast<unsigned long long>(y)+0x80000000)»32);
#else
    return mulhi<std::round_to_nearest>(x, y);
#endif
}

inline uint32 divide64(uint32 x, uint32 y, int &s)
{
#if HALF_ENABLE_CPP11_LONG_LONG
    unsigned long long xx = static_cast<unsigned long long>(x) « 32;
    return s = (xx%y!=0), static_cast<uint32>(xx/y);
#else
    y »= 1;
    uint32 rem = x, div = 0;
    for(unsigned int i=0; i<32; ++i)
    {
        div «= 1;
        if(rem >= y)
        {
            rem -= y;
            div |= 1;
        }
        rem «= 1;
    }
    return s = rem > 1, div;
#endif
}

template<bool Q,bool R> unsigned int mod(unsigned int x, unsigned int y, int *quo = NULL)
{
    unsigned int q = 0;
    if(x > y)
    {
        int absx = x, absy = y, expx = 0, expy = 0;
        for(; absx<0x400; absx«=1,--expx) ;
        for(; absy<0x400; absy«=1,--expy) ;
        expx += absx » 10;
        expy += absy » 10;
        int mx = (absx&0x3FF) | 0x400, my = (absy&0x3FF) | 0x400;
        for(int d=expx-expy; d; --d)
        {
            if(!Q && mx == my)
                return 0;
            if(mx >= my)
            {
                mx -= my;
                q += Q;
            }
            mx «= 1;
            q «= static_cast<int>(Q);
        }
        if(!Q && mx == my)
            return 0;
        if(mx >= my)
        {
            mx -= my;
            ++q;
        }
        if(Q)
        {
            q &= (1«(std::numeric_limits<int>::digits-1)) - 1;
            if(!mx)
                return *quo = q, 0;
        }
        for(; mx<0x400; mx«=1,--expy) ;
        x = (expy>0) ? ((expy«10)|(mx&0x3FF)) : (mx»(1-expy));
    }
    if(R)
    {
        unsigned int a, b;
        if(y < 0x800)
        {
            a = (x<0x400) ? (x«1) : (x+0x400);
            b = y;
        }
        else
        {
            a = x;
            b = y - 0x400;
        }
        if(a > b || (a == b && (q&1)))
        {
            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));
            for(; m<0x800 && exp>1; m«=1,--exp) ;
            x = 0x8000 + ((exp-1)«10) + (m»1);
            q += Q;
        }
    }
    if(Q)
        *quo = q;
    return x;
}

template<unsigned int F> uint32 sqrt(uint32 &r, int &exp)
{
    int i = exp & 1;
    r «= i;
    exp = (exp-i) / 2;
    uint32 m = 0;
    for(uint32 bit=static_cast<uint32>(1)«F; bit; bit»=2)
    {
        if(r < m+bit)
            m »= 1;
        else
        {
            r -= m + bit;
            m = (m»1) + bit;
        }
    }
    return m;
}

inline uint32 exp2(uint32 m, unsigned int n = 32)
{
    static const uint32 logs[] = {
        0x80000000, 0x4AE00D1D, 0x2934F098, 0x15C01A3A, 0x0B31FB7D, 0x05AEB4DD, 0x02DCF2D1, 0x016FE50B,
        0x00B84E23, 0x005C3E10, 0x002E24CA, 0x001713D6, 0x000B8A47, 0x0005C53B, 0x0002E2A3, 0x00017153,
        0x0000B8AA, 0x00005C55, 0x00002E2B, 0x00001715, 0x00000B8B, 0x000005C5, 0x000002E3, 0x00000171,
        0x000000B9, 0x0000005C, 0x0000002E, 0x00000017, 0x0000000C, 0x00000006, 0x00000003, 0x00000001 };
    if(!m)
        return 0x80000000;
    uint32 mx = 0x80000000, my = 0;
    for(unsigned int i=1; i<n; ++i)
    {
        uint32 mz = my + logs[i];
        if(mz <= m)
        {
            my = mz;
            mx += mx » i;
        }
    }
    return mx;
}

inline uint32 log2(uint32 m, unsigned int n = 32)
{
    static const uint32 logs[] = {
        0x80000000, 0x4AE00D1D, 0x2934F098, 0x15C01A3A, 0x0B31FB7D, 0x05AEB4DD, 0x02DCF2D1, 0x016FE50B,
        0x00B84E23, 0x005C3E10, 0x002E24CA, 0x001713D6, 0x000B8A47, 0x0005C53B, 0x0002E2A3, 0x00017153,
        0x0000B8AA, 0x00005C55, 0x00002E2B, 0x00001715, 0x00000B8B, 0x000005C5, 0x000002E3, 0x00000171,
        0x000000B9, 0x0000005C, 0x0000002E, 0x00000017, 0x0000000C, 0x00000006, 0x00000003, 0x00000001 };
    if(m == 0x40000000)
        return 0;
    uint32 mx = 0x40000000, my = 0;
    for(unsigned int i=1; i<n; ++i)
    {
        uint32 mz = mx + (mx»i);
        if(mz <= m)
        {
            mx = mz;
            my += logs[i];
        }
    }
    return my;
}

inline std::pair<uint32,uint32> sincos(uint32 mz, unsigned int n = 31)
{
    static const uint32 angles[] = {
        0x3243F6A9, 0x1DAC6705, 0x0FADBAFD, 0x07F56EA7, 0x03FEAB77, 0x01FFD55C, 0x00FFFAAB, 0x007FFF55,
        0x003FFFEB, 0x001FFFFD, 0x00100000, 0x00080000, 0x00040000, 0x00020000, 0x00010000, 0x00008000,
        0x00004000, 0x00002000, 0x00001000, 0x00000800, 0x00000400, 0x00000200, 0x00000100, 0x00000080,
        0x00000040, 0x00000020, 0x00000010, 0x00000008, 0x00000004, 0x00000002, 0x00000001 };
    uint32 mx = 0x26DD3B6A, my = 0;
    for(unsigned int i=0; i<n; ++i)
    {
        uint32 sign = sign_mask(mz);
        uint32 tx = mx - (arithmetic_shift(my, i)^sign) + sign;
        uint32 ty = my + (arithmetic_shift(mx, i)^sign) - sign;
        mx = tx; my = ty; mz -= (angles[i]^sign) - sign;
    }
    return std::make_pair(my, mx);
}

inline uint32 atan2(uint32 my, uint32 mx, unsigned int n = 31)
{
    static const uint32 angles[] = {
        0x3243F6A9, 0x1DAC6705, 0x0FADBAFD, 0x07F56EA7, 0x03FEAB77, 0x01FFD55C, 0x00FFFAAB, 0x007FFF55,
        0x003FFFEB, 0x001FFFFD, 0x00100000, 0x00080000, 0x00040000, 0x00020000, 0x00010000, 0x00008000,
        0x00004000, 0x00002000, 0x00001000, 0x00000800, 0x00000400, 0x00000200, 0x00000100, 0x00000080,
        0x00000040, 0x00000020, 0x00000010, 0x00000008, 0x00000004, 0x00000002, 0x00000001 };
    uint32 mz = 0;
    for(unsigned int i=0; i<n; ++i)
    {
        uint32 sign = sign_mask(my);
        uint32 tx = mx + (arithmetic_shift(my, i)^sign) - sign;
        uint32 ty = my - (arithmetic_shift(mx, i)^sign) + sign;
        mx = tx; my = ty; mz += (angles[i]^sign) - sign;
    }
    return mz;
}

inline uint32 angle_arg(unsigned int abs, int &k)
{
    uint32 m = (abs&0x3FF) | ((abs>0x3FF)«10);
    int exp = (abs»10) + (abs<=0x3FF) - 15;
    if(abs < 0x3A48)
        return k = 0, m « (exp+20);
#if HALF_ENABLE_CPP11_LONG_LONG
    unsigned long long y = m * 0xA2F9836E4E442, mask = (1ULL«(62-exp)) - 1, yi = (y+(mask»1)) & ~mask, f = y - yi;
    uint32 sign = -static_cast<uint32>(f»63);
    k = static_cast<int>(yi»(62-exp));
    return (multiply64(static_cast<uint32>((sign ? -f : f)»(31-exp)), 0xC90FDAA2)^sign) - sign;
#else
    uint32 yh = m*0xA2F98 + mulhi<std::round_toward_zero>(m, 0x36E4E442), yl = (m*0x36E4E442) & 0xFFFFFFFF;
    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;
    return (multiply64((exp>-1) ? (((fh«(1+exp))&0xFFFFFFFF)|((fl&0xFFFFFFFF)»(31-exp))) : fh, 0xC90FDAA2)^sign) - sign;
#endif
}

inline std::pair<uint32,uint32> atan2_args(unsigned int abs)
{
    int exp = -15;
    for(; abs<0x400; abs«=1,--exp) ;
    exp += abs » 10;
    uint32 my = ((abs&0x3FF)|0x400) « 5, r = my * my;
    int rexp = 2 * exp;
    r = 0x40000000 - ((rexp>-31) ? ((r»-rexp)|((r&((static_cast<uint32>(1)«-rexp)-1))!=0)) : 1);
    for(rexp=0; r<0x40000000; r«=1,--rexp) ;
    uint32 mx = sqrt<30>(r, rexp);
    int d = exp - rexp;
    if(d < 0)
        return std::make_pair((d<-14) ? ((my»(-d-14))+((my»(-d-15))&1)) : (my«(14+d)), (mx«14)+(r«13)/mx);
    if(d > 0)
        return std::make_pair(my«14, (d>14) ? ((mx»(d-14))+((mx»(d-15))&1)) : ((d==14) ? mx : ((mx«(14-d))+(r«(13-d))/mx)));
    return std::make_pair(my«13, (mx«13)+(r«12)/mx);
}

inline std::pair<uint32,uint32> hyperbolic_args(unsigned int abs, int &exp, unsigned int n = 32)
{
    uint32 mx = detail::multiply64(static_cast<uint32>((abs&0x3FF)+((abs>0x3FF)«10))«21, 0xB8AA3B29), my;
    int e = (abs»10) + (abs<=0x3FF);
    if(e < 14)
    {
        exp = 0;
        mx »= 14 - e;
    }
    else
    {
        exp = mx » (45-e);
        mx = (mx«(e-14)) & 0x7FFFFFFF;
    }
    mx = exp2(mx, n);
    int d = exp « 1, s;
    if(mx > 0x80000000)
    {
        my = divide64(0x80000000, mx, s);
        my |= s;
        ++d;
    }
    else
        my = mx;
    return std::make_pair(mx, (d<31) ? ((my»d)|((my&((static_cast<uint32>(1)«d)-1))!=0)) : 1);
}

template<std::float_round_style R> unsigned int exp2_post(uint32 m, int exp, bool esign, unsigned int sign = 0, unsigned int n = 32)
{
    if(esign)
    {
        exp = -exp - (m!=0);
        if(exp < -25)
            return underflow<R>(sign);
        else if(exp == -25)
            return rounded<R,false>(sign, 1, m!=0);
    }
    else if(exp > 15)
        return overflow<R>(sign);
    if(!m)
        return sign | (((exp+=15)>0) ? (exp«10) : check_underflow(0x200»-exp));
    m = exp2(m, n);
    int s = 0;
    if(esign)
        m = divide64(0x80000000, m, s);
    return fixed2half<R,31,false,false,true>(m, exp+14, sign, s);
}

template<std::float_round_style R,uint32 L> unsigned int log2_post(uint32 m, int ilog, int exp, unsigned int sign = 0)
{
    uint32 msign = sign_mask(ilog);
    m = (((static_cast<uint32>(ilog)«27)+(m»4))^msign) - msign;
    if(!m)
        return 0;
    for(; m<0x80000000; m«=1,--exp) ;
    int i = m >= L, s;
    exp += i;
    m »= 1 + i;
    sign ^= msign & 0x8000;
    if(exp < -11)
        return underflow<R>(sign);
    m = divide64(m, L, s);
    return fixed2half<R,30,false,false,true>(m, exp, sign, 1);
```

```
}

template<std::float_round_style R> unsigned int hypot_post(uint32 r, int exp)
{
    int i = r » 31;
    if((exp+=i) > 46)
        return overflow<R>();
    if(exp < -34)
        return underflow<R>();
    r = (r»i) | (r&i);
    uint32 m = sqrt<30>(r, exp+=15);
    return fixed2half<R,15,false,false,false>(m, exp-1, 0, r!=0);
}

template<std::float_round_style R> unsigned int tangent_post(uint32 my, uint32 mx, int exp, unsigned int sign = 0)
{
    int i = my >= mx, s;
    exp += i;
    if(exp > 29)
        return overflow<R>(sign);
    if(exp < -11)
        return underflow<R>(sign);
    uint32 m = divide64(my»(i+1), mx, s);
    return fixed2half<R,30,false,false,true>(m, exp, sign, s);
}

template<std::float_round_style R,bool S> unsigned int area(unsigned int arg)
{
    int abs = arg & 0x7FFF, expx = (abs»10) + (abs<=0x3FF) - 15, expy = -15, ilog, i;
    uint32 mx = static_cast<uint32>((abs&0x3FF)|((abs>0x3FF)«10)) « 20, my, r;
    for(; abs<0x400; abs«=1,--expy) ;
    expy += abs » 10;
    r = ((abs&0x3FF)|0x400) « 5;
    r *= r;
    i = r » 31;
    expy = 2*expy + i;
    r »= i;
    if(S)
    {
        if(expy < 0)
        {
            r = 0x40000000 + ((expy>-30) ? ((r»-expy)|((r&((static_cast<uint32>(1)«-expy)-1))!=0)) : 1);
            expy = 0;
        }
        else
        {
            r += 0x40000000 » expy;
            i = r » 31;
            r = (r»i) | (r&i);
            expy += i;
        }
    }
    else
    {
        r -= 0x40000000 » expy;
        for(; r<0x40000000; r«=1,--expy) ;
    }
    my = sqrt<30>(r, expy);
    my = (my«15) + (r«14)/my;
    if(S)
    {
        mx »= expy - expx;
        ilog = expy;
    }
    else
    {
        my »= expx - expy;
        ilog = expx;
    }
    my += mx;
    i = my » 31;
    static const int G = S && (R==std::round_to_nearest);
    return log2_post<R,0xB8AA3B2A>(log2(my»i, 26+S+G)+(G«3), ilog+i, 17, arg&(static_cast<unsigned>(S)«15));
}

struct f31
{
    HALF_CONSTEXPR f31(uint32 mant, int e) : m(mant), exp(e) {}

    f31(unsigned int abs) : exp(-15)
    {
        for(; abs<0x400; abs«=1,--exp) ;
        m = static_cast<uint32>((abs&0x3FF)|0x400) « 21;
        exp += (abs»10);
```

```
            }

            friend f31 operator+(f31 a, f31 b)
            {
                if(b.exp > a.exp)
                    std::swap(a, b);
                int d = a.exp - b.exp;
                uint32 m = a.m + ((d<32) ? (b.m»d) : 0);
                int i = (m&0xFFFFFFFF) < a.m;
                return f31(((m+i)»i)|0x80000000, a.exp+i);
            }

            friend f31 operator-(f31 a, f31 b)
            {
                int d = a.exp - b.exp, exp = a.exp;
                uint32 m = a.m - ((d<32) ? (b.m»d) : 0);
                if(!m)
                    return f31(0, -32);
                for(; m<0x80000000; m«=1,--exp) ;
                return f31(m, exp);
            }

            friend f31 operator*(f31 a, f31 b)
            {
                uint32 m = multiply64(a.m, b.m);
                int i = m » 31;
                return f31(m«(1-i), a.exp + b.exp + i);
            }

            friend f31 operator/(f31 a, f31 b)
            {
                int i = a.m >= b.m, s;
                uint32 m = divide64((a.m+i)»i, b.m, s);
                return f31(m, a.exp - b.exp + i - 1);
            }

            uint32 m;
            int exp;
        };

        template<std::float_round_style R,bool C> unsigned int erf(unsigned int arg)
        {
            unsigned int abs = arg & 0x7FFF, sign = arg & 0x8000;
            f31 x(abs), x2 = x * x * f31(0xB8AA3B29, 0), t = f31(0x80000000, 0) / (f31(0x80000000, 0)+f31(0xA7BA054A, -2)*x), t2 = t * t;
            f31 e = ((f31(0x87DC2213, 0)*t2+f31(0xB5F0E2AE, 0))*t2+f31(0x82790637, -2)-(f31(0xBA00E2B8, 0)*t2+f31(0x91A98E62, -2))*t) * t /
                    ((x2.exp<0) ? f31(exp2((x2.exp>-32) ? (x2.m»-x2.exp) : 0, 30), 0) : f31(exp2((x2.m«x2.exp)&0x7FFFFFFF, 22), x2.m»(31-x2.exp)));
            return (!C || sign) ? fixed2half<R,31,false,true,true>(0x80000000-(e.m»(C-e.exp)), 14+C, sign&(C-1U)) :
                    (e.exp<-25) ? underflow<R>() : fixed2half<R,30,false,false,true>(e.m»1, e.exp+14, 0, e.m&1);
        }

        template<std::float_round_style R,bool L> unsigned int gamma(unsigned int arg)
        {
/*          static const double p[] ={ 2.50662827563479526904, 225.525584619175212544, -268.295973841304927459, 80.9030806934622512966, -5.00757863970517583837, 0.0114684895434781459556 };
            double t = arg + 4.65, s = p[0];
            for(unsigned int i=0; i<5; ++i)
                s += p[i+1] / (arg+i);
            return std::log(s) + (arg-0.5)*std::log(t) - t;
*/          static const f31 pi(0xC90FDAA2, 1), lbe(0xB8AA3B29, 0);
            unsigned int abs = arg & 0x7FFF, sign = arg & 0x8000;
            bool bsign = sign != 0;
            f31 z(abs), x = sign ? (z+f31(0x80000000, 0)) : z, t = x + f31(0x94CCCCCD, 2), s =
                f31(0xA06C9901, 1) + f31(0xBBE654E2, -7)/(x+f31(0x80000000, 2)) + f31(0xA1CE6098, 6)/(x+f31(0x80000000, 1))
                  + f31(0xE1868CB7, 7)/x - f31(0x8625E279, 8)/(x+f31(0x80000000, 0)) - f31(0xA03E158F, 2)/(x+f31(0xC0000000, 1));
            int i = (s.exp>=2) + (s.exp>=4) + (s.exp>=8) + (s.exp>=16);
            s = f31((static_cast<uint32>(s.exp)«(31-i))+(log2(s.m»1, 28)»i), i) / lbe;
            if(x.exp != -1 || x.m != 0x80000000)
            {
                i = (t.exp>=2) + (t.exp>=4) + (t.exp>=8);
                f31 l = f31((static_cast<uint32>(t.exp)«(31-i))+(log2(t.m»1, 30)»i), i) / lbe;
                s = (x.exp<-1) ? (s-(f31(0x80000000, -1)-x)*l) : (s+(x-f31(0x80000000, -1))*l);
            }
            s = x.exp ? (s-t) : (t-s);
            if(bsign)
            {
                if(z.exp >= 0)
                {
                    sign &= (L|((z.m»(31-z.exp))&1)) - 1;
                    for(z=f31((z.m«(1+z.exp))&0xFFFFFFFF, -1); z.m<0x80000000; z.m«=1,--z.exp) ;
                }
                if(z.exp == -1)
                    z = f31(0x80000000, 0) - z;
                if(z.exp < -1)
                {
                    z = z * pi;
                    z.m = sincos(z.m»(1-z.exp), 30).first;
                    for(z.exp=1; z.m<0x80000000; z.m«=1,--z.exp) ;
                }
                else
                    z = f31(0x80000000, 0);
            }
            if(L)
            {
                if(bsign)
                {
                    f31 l(0x92868247, 0);
                    if(z.exp < 0)
                    {
                        uint32 m = log2((z.m+1)»1, 27);
                        z = f31(-((static_cast<uint32>(z.exp)«26)+(m»5)), 5);
                        for(; z.m<0x80000000; z.m«=1,--z.exp) ;
                        l = l + z / lbe;
                    }
                    sign = static_cast<unsigned>(x.exp&&(l.exp<s.exp||(l.exp==s.exp&&l.m<s.m))) « 15;
                    s = sign ? (s-l) : x.exp ? (l-s) : (l+s);
                }
                else
                {
                    sign = static_cast<unsigned>(x.exp==0) « 15;
                    if(s.exp < -24)
                        return underflow<R>(sign);
                    if(s.exp > 15)
                        return overflow<R>(sign);
                }
            }
            else
            {
                s = s * lbe;
                uint32 m;
                if(s.exp < 0)
                {
                    m = s.m » -s.exp;
                    s.exp = 0;
                }
                else
                {
                    m = (s.m«s.exp) & 0x7FFFFFFF;
                    s.exp = (s.m»(31-s.exp));
                }
                s.m = exp2(m, 27);
                if(!x.exp)
                    s = f31(0x80000000, 0) / s;
                if(bsign)
                {
                    if(z.exp < 0)
                        s = s * z;
                    s = pi / s;
                    if(s.exp < -24)
                        return underflow<R>(sign);
                }
                else if(z.exp > 0 && !(z.m&((1«(31-z.exp))-1)))
                    return ((s.exp+14)«10) + (s.m»21);
                if(s.exp > 15)
                    return overflow<R>(sign);
            }
            return fixed2half<R,31,false,false,true>(s.m, s.exp+14, sign);
        }

        template<typename,typename,std::float_round_style> struct half_caster;
    }

    class half
    {
    public:

        HALF_CONSTEXPR half() HALF_NOEXCEPT : data_() {}

        explicit half(float rhs) :
data_(static_cast<detail::uint16>(detail::float2half<round_style>(rhs))) {}

        operator float() const { return detail::half2float<float>(data_); }

        half& operator=(float rhs) { data_ =
static_cast<detail::uint16>(detail::float2half<round_style>(rhs)); return *this; }


        half& operator+=(half rhs) { return *this = *this + rhs; }
        half& operator-=(half rhs) { return *this = *this - rhs; }

        half& operator*=(half rhs) { return *this = *this * rhs; }

        half& operator/=(half rhs) { return *this = *this / rhs; }

        half& operator+=(float rhs) { return *this = *this + rhs; }

        half& operator-=(float rhs) { return *this = *this - rhs; }

        half& operator*=(float rhs) { return *this = *this * rhs; }

        half& operator/=(float rhs) { return *this = *this / rhs; }


        half& operator++() { return *this = *this + half(detail::binary, 0x3C00); }

        half& operator--() { return *this = *this + half(detail::binary, 0xBC00); }

        half operator++(int) { half out(*this); ++*this; return out; }

        half operator--(int) { half out(*this); --*this; return out; }

    private:
        static const std::float_round_style round_style = (std::float_round_style)(HALF_ROUND_STYLE);

        HALF_CONSTEXPR half(detail::binary_t, unsigned int bits) HALF_NOEXCEPT : data_(static_cast<detail::uint16>(bits)) {}

        detail::uint16 data_;

    #ifndef HALF_DOXYGEN_ONLY
        friend HALF_CONSTEXPR_NOERR bool operator==(half, half);
        friend HALF_CONSTEXPR_NOERR bool operator!=(half, half);
        friend HALF_CONSTEXPR_NOERR bool operator<(half, half);
        friend HALF_CONSTEXPR_NOERR bool operator>(half, half);
        friend HALF_CONSTEXPR_NOERR bool operator<=(half, half);
        friend HALF_CONSTEXPR_NOERR bool operator>=(half, half);
        friend HALF_CONSTEXPR half operator-(half);
        friend half operator+(half, half);
        friend half operator-(half, half);
        friend half operator*(half, half);
        friend half operator/(half, half);
        template<typename charT,typename traits> friend std::basic_ostream<charT,traits>& operator«(std::basic_ostream<charT,traits>&, half);
        template<typename charT,typename traits> friend std::basic_istream<charT,traits>& operator»(std::basic_istream<charT,traits>&, half&);
        friend HALF_CONSTEXPR half fabs(half);
        friend half fmod(half, half);
        friend half remainder(half, half);
        friend half remquo(half, half, int*);
        friend half fma(half, half, half);
        friend HALF_CONSTEXPR_NOERR half fmax(half, half);
        friend HALF_CONSTEXPR_NOERR half fmin(half, half);
        friend half fdim(half, half);
        friend half nanh(const char*);
        friend half exp(half);
        friend half exp2(half);
        friend half expm1(half);
        friend half log(half);
        friend half log10(half);
        friend half log2(half);
        friend half log1p(half);
        friend half sqrt(half);
        friend half rsqrt(half);
        friend half cbrt(half);
        friend half hypot(half, half);
        friend half hypot(half, half, half);
        friend half pow(half, half);
        friend void sincos(half, half*, half*);
        friend half sin(half);
        friend half cos(half);
        friend half tan(half);
        friend half asin(half);
        friend half acos(half);
        friend half atan(half);
        friend half atan2(half, half);
        friend half sinh(half);
        friend half cosh(half);
        friend half tanh(half);
        friend half asinh(half);
        friend half acosh(half);
        friend half atanh(half);
        friend half erf(half);
        friend half erfc(half);
        friend half lgamma(half);
        friend half tgamma(half);
        friend half ceil(half);
        friend half floor(half);
        friend half trunc(half);
        friend half round(half);
        friend long lround(half);
        friend half rint(half);
        friend long lrint(half);
        friend half nearbyint(half);
    #ifdef HALF_ENABLE_CPP11_LONG_LONG
        friend long long llround(half);
        friend long long llrint(half);
    #endif
        friend half frexp(half, int*);
        friend half scalbln(half, long);
        friend half modf(half, half*);
        friend int ilogb(half);
        friend half logb(half);
        friend half nextafter(half, half);
        friend half nexttoward(half, long double);
        friend HALF_CONSTEXPR half copysign(half, half);
        friend HALF_CONSTEXPR int fpclassify(half);
        friend HALF_CONSTEXPR bool isfinite(half);
        friend HALF_CONSTEXPR bool isinf(half);
        friend HALF_CONSTEXPR bool isnan(half);
        friend HALF_CONSTEXPR bool isnormal(half);
        friend HALF_CONSTEXPR bool signbit(half);
        friend HALF_CONSTEXPR bool isgreater(half, half);
        friend HALF_CONSTEXPR bool isgreaterequal(half, half);
        friend HALF_CONSTEXPR bool isless(half, half);
        friend HALF_CONSTEXPR bool islessequal(half, half);
        friend HALF_CONSTEXPR bool islessgreater(half, half);
        template<typename,typename,std::float_round_style> friend struct detail::half_caster;
        friend class std::numeric_limits<half>;
    #if HALF_ENABLE_CPP11_HASH
        friend struct std::hash<half>;
    #endif
    #if HALF_ENABLE_CPP11_USER_LITERALS
        friend half literal::operator "" _h(long double);
    #endif
    #endif
    };

#if HALF_ENABLE_CPP11_USER_LITERALS
    namespace literal
    {
        inline half operator "" _h(long double value) { return half(detail::binary, detail::float2half<half::round_style>(value)); }
    }
#endif

    namespace detail
    {
        template<typename T,typename U,std::float_round_style R=(std::float_round_style)(HALF_ROUND_STYLE)> struct half_caster {};
        template<typename U,std::float_round_style R> struct half_caster<half,U,R>
        {
        #if HALF_ENABLE_CPP11_STATIC_ASSERT && HALF_ENABLE_CPP11_TYPE_TRAITS
            static_assert(std::is_arithmetic<U>::value, "half_cast from non-arithmetic type unsupported");
        #endif

            static half cast(U arg) { return cast_impl(arg, is_float<U>()); };

        private:
            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)); }
        };
        template<typename T,std::float_round_style R> struct half_caster<T,half,R>
        {
        #if HALF_ENABLE_CPP11_STATIC_ASSERT && HALF_ENABLE_CPP11_TYPE_TRAITS
            static_assert(std::is_arithmetic<T>::value, "half_cast to non-arithmetic type unsupported");
        #endif

            static T cast(half arg) { return cast_impl(arg, is_float<T>()); }

        private:
            static T cast_impl(half arg, true_type) { return half2float<T>(arg.data_); }
            static T cast_impl(half arg, false_type) { return half2int<R,true,true,T>(arg.data_); }
        };
        template<std::float_round_style R> struct half_caster<half,half,R>
        {
            static half cast(half arg) { return arg; }
        };
    }
}
```

```

namespace std
{
    template<> class numeric_limits<half_float::half>
    {
    public:
        static HALF_CONSTEXPR_CONST bool is_specialized = true;

        static HALF_CONSTEXPR_CONST bool is_signed = true;

        static HALF_CONSTEXPR_CONST bool is_integer = false;

        static HALF_CONSTEXPR_CONST bool is_exact = false;

        static HALF_CONSTEXPR_CONST bool is_modulo = false;

        static HALF_CONSTEXPR_CONST bool is_bounded = true;

        static HALF_CONSTEXPR_CONST bool is_iec559 = true;

        static HALF_CONSTEXPR_CONST bool has_infinity = true;

        static HALF_CONSTEXPR_CONST bool has_quiet_NaN = true;

        static HALF_CONSTEXPR_CONST bool has_signaling_NaN = true;

        static HALF_CONSTEXPR_CONST float_denorm_style has_denorm = denorm_present;

        static HALF_CONSTEXPR_CONST bool has_denorm_loss = false;

    #if HALF_ERRHANDLING_THROWS
        static HALF_CONSTEXPR_CONST bool traps = true;
    #else
        static HALF_CONSTEXPR_CONST bool traps = false;
    #endif

        static HALF_CONSTEXPR_CONST bool tinyness_before = false;

        static HALF_CONSTEXPR_CONST float_round_style round_style = half_float::half::round_style;

        static HALF_CONSTEXPR_CONST int digits = 11;

        static HALF_CONSTEXPR_CONST int digits10 = 3;

        static HALF_CONSTEXPR_CONST int max_digits10 = 5;

        static HALF_CONSTEXPR_CONST int radix = 2;

        static HALF_CONSTEXPR_CONST int min_exponent = -13;

        static HALF_CONSTEXPR_CONST int min_exponent10 = -4;

        static HALF_CONSTEXPR_CONST int max_exponent = 16;

        static HALF_CONSTEXPR_CONST int max_exponent10 = 4;

        static HALF_CONSTEXPR half_float::half min() HALF_NOTHROW { return half_float::half(half_float::detail::binary, 0x0400); }

        static HALF_CONSTEXPR half_float::half lowest() HALF_NOTHROW { return half_float::half(half_float::detail::binary, 0xFBFF); }

        static HALF_CONSTEXPR half_float::half max() HALF_NOTHROW { return half_float::half(half_float::detail::binary, 0x7BFF); }

        static HALF_CONSTEXPR half_float::half epsilon() HALF_NOTHROW { return half_float::half(half_float::detail::binary, 0x1400); }

        static HALF_CONSTEXPR half_float::half round_error() HALF_NOTHROW
            { return half_float::half(half_float::detail::binary, (round_style==std::round_to_nearest) ? 0x3800 : 0x3C00); }

        static HALF_CONSTEXPR half_float::half infinity() HALF_NOTHROW { return half_float::half(half_float::detail::binary, 0x7C00); }

        static HALF_CONSTEXPR half_float::half quiet_NaN() HALF_NOTHROW { return half_float::half(half_float::detail::binary, 0x7FFF); }

        static HALF_CONSTEXPR half_float::half signaling_NaN() HALF_NOTHROW { return half_float::half(half_float::detail::binary, 0x7DFF); }

        static HALF_CONSTEXPR half_float::half denorm_min() HALF_NOTHROW { return half_float::half(half_float::detail::binary, 0x0001); }
    };

#if HALF_ENABLE_CPP11_HASH
    template<> struct hash<half_float::half>
    {
        typedef half_float::half argument_type;

        typedef size_t result_type;

        result_type operator()(argument_type arg) const { return hash<half_float::detail::uint16>()(arg.data_&-static_cast<unsigned>(arg.data_!=0x8000)); }
    };
#endif
}

namespace half_float
{

    inline HALF_CONSTEXPR_NOERR bool operator==(half x, half y)
    {
        return !detail::compsignal(x.data_, y.data_) && (x.data_==y.data_ || !((x.data_|y.data_)&0x7FFF));
    }

    inline HALF_CONSTEXPR_NOERR bool operator!=(half x, half y)
    {
        return detail::compsignal(x.data_, y.data_) || (x.data_!=y.data_ && ((x.data_|y.data_)&0x7FFF));
    }

    inline HALF_CONSTEXPR_NOERR bool operator<(half x, half y)
    {
        return !detail::compsignal(x.data_, y.data_) &&
            ((x.data_^(0x8000|(0x8000-(x.data_»15))))+(x.data_»15)) < ((y.data_^(0x8000|(0x8000-(y.data_»15))))+(y.data_»15));
    }

    inline HALF_CONSTEXPR_NOERR bool operator>(half x, half y)
    {
        return !detail::compsignal(x.data_, y.data_) &&
            ((x.data_^(0x8000|(0x8000-(x.data_»15))))+(x.data_»15)) > ((y.data_^(0x8000|(0x8000-(y.data_»15))))+(y.data_»15));
    }

    inline HALF_CONSTEXPR_NOERR bool operator<=(half x, half y)
    {
        return !detail::compsignal(x.data_, y.data_) &&
            ((x.data_^(0x8000|(0x8000-(x.data_»15))))+(x.data_»15)) <= ((y.data_^(0x8000|(0x8000-(y.data_»15))))+(y.data_»15));
    }

    inline HALF_CONSTEXPR_NOERR bool operator>=(half x, half y)
    {
        return !detail::compsignal(x.data_, y.data_) &&
            ((x.data_^(0x8000|(0x8000-(x.data_»15))))+(x.data_»15)) >= ((y.data_^(0x8000|(0x8000-(y.data_»15))))+(y.data_»15));
    }



    inline HALF_CONSTEXPR half operator+(half arg) { return arg; }

    inline HALF_CONSTEXPR half operator-(half arg) { return half(detail::binary, arg.data_^0x8000); }

    inline half operator+(half x, half y)
    {
    #ifdef HALF_ARITHMETIC_TYPE
        return half(detail::binary, detail::float2half<half::round_style>(detail::half2float<detail::internal_t>(x.data_)+detail::half2float<detail::interna
    #else
        int absx = x.data_ & 0x7FFF, absy = y.data_ & 0x7FFF;
        bool sub = ((x.data_^y.data_)&0x8000) != 0;
        if(absx >= 0x7C00 || absy >= 0x7C00)
            return half(detail::binary,   (absx>0x7C00 || absy>0x7C00) ? detail::signal(x.data_, y.data_) : (absy!=0x7C00) ? x.data_ :
                                        (sub && absx==0x7C00) ? detail::invalid() : y.data_);
        if(!absx)
            return absy ? y : half(detail::binary, (half::round_style==std::round_toward_neg_infinity) ? (x.data_|y.data_) : (x.data_&y.data_));
        if(!absy)
            return x;
        unsigned int sign = ((sub && absy>absx) ? y.data_ : x.data_) & 0x8000;
        if(absy > absx)
            std::swap(absx, absy);
        int exp = (absx»10) + (absx<=0x3FF), d = exp - (absy»10) - (absy<=0x3FF), mx = ((absx&0x3FF)|((absx>0x3FF)«10)) « 3, my;
        if(d < 13)
        {
            my = ((absy&0x3FF)|((absy>0x3FF)«10)) « 3;
            my = (my»d) | ((my&((1«d)-1))!=0);
        }
        else
```

```
            my = 1;
        if(sub)
        {
            if(!(mx-=my))
                return half(detail::binary,
static_cast<unsigned>(half::round_style==std::round_toward_neg_infinity)«15);
            for(; mx<0x2000 && exp>1; mx«=1,--exp) ;
        }
        else
        {
            mx += my;
            int i = mx » 14;
            if((exp+=i) > 30)
                return half(detail::binary, detail::overflow<half::round_style>(sign));
            mx = (mx»i) | (mx&i);
        }
        return half(detail::binary, detail::rounded<half::round_style,false>(sign+((exp-1)«10)+(mx»3),
(mx»2)&1, (mx&0x3)!=0));
    #endif
    }

    inline half operator-(half x, half y)
    {
    #ifdef HALF_ARITHMETIC_TYPE
        return half(detail::binary,
detail::float2half<half::round_style>(detail::half2float<detail::internal_t>(x.data_)-detail::half2float<detail::interna
    #else
        return x + -y;
    #endif
    }

    inline half operator*(half x, half y)
    {
    #ifdef HALF_ARITHMETIC_TYPE
        return half(detail::binary,
detail::float2half<half::round_style>(detail::half2float<detail::internal_t>(x.data_)*detail::half2float<detail::interna
    #else
        int absx = x.data_ & 0x7FFF, absy = y.data_ & 0x7FFF, exp = -16;
        unsigned int sign = (x.data_^y.data_) & 0x8000;
        if(absx >= 0x7C00 || absy >= 0x7C00)
            return half(detail::binary,    (absx>0x7C00 || absy>0x7C00) ? detail::signal(x.data_,
y.data_) :
                                        ((absx==0x7C00 && !absy)||(absy==0x7C00 && !absx)) ?
detail::invalid() : (sign|0x7C00));
        if(!absx || !absy)
            return half(detail::binary, sign);
        for(; absx<0x400; absx«=1,--exp) ;
        for(; absy<0x400; absy«=1,--exp) ;
        detail::uint32 m = static_cast<detail::uint32>((absx&0x3FF)|0x400) *
static_cast<detail::uint32>((absy&0x3FF)|0x400);
        int i = m » 21, s = m & i;
        exp += (absx»10) + (absy»10) + i;
        if(exp > 29)
            return half(detail::binary, detail::overflow<half::round_style>(sign));
        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,
exp, sign, s));
    #endif
    }

    inline half operator/(half x, half y)
    {
    #ifdef HALF_ARITHMETIC_TYPE
        return half(detail::binary,
detail::float2half<half::round_style>(detail::half2float<detail::internal_t>(x.data_)/detail::half2float<detail::interna
    #else
        int absx = x.data_ & 0x7FFF, absy = y.data_ & 0x7FFF, exp = 14;
        unsigned int sign = (x.data_^y.data_) & 0x8000;
        if(absx >= 0x7C00 || absy >= 0x7C00)
            return half(detail::binary,    (absx>0x7C00 || absy>0x7C00) ? detail::signal(x.data_,
y.data_) :
                                        (absx==absy) ? detail::invalid() : (sign|((absx==0x7C00) ?
0x7C00 : 0)));
        if(!absx)
            return half(detail::binary, absy ? sign : detail::invalid());
        if(!absy)
            return half(detail::binary, detail::pole(sign));
        for(; absx<0x400; absx«=1,--exp) ;
        for(; absy<0x400; absy«=1,++exp) ;
        detail::uint32 mx = (absx&0x3FF) | 0x400, my = (absy&0x3FF) | 0x400;
        int i = mx < my;
        exp += (absx»10) - (absy»10) - i;
        if(exp > 29)
            return half(detail::binary, detail::overflow<half::round_style>(sign));
        else if(exp < -11)
            return half(detail::binary, detail::underflow<half::round_style>(sign));
        mx «= 12 + i;
        my «= 1;
        return half(detail::binary, detail::fixed2half<half::round_style,11,false,false,false>(mx/my, exp, sign, mx%my!=0));
    #endif
    }



    template<typename charT,typename traits> std::basic_ostream<charT,traits>& operator«(std::basic_ostream<charT,traits> &out, half arg)
    {
    #ifdef HALF_ARITHMETIC_TYPE
        return out « detail::half2float<detail::internal_t>(arg.data_);
    #else
        return out « detail::half2float<float>(arg.data_);
    #endif
    }

    template<typename charT,typename traits> std::basic_istream<charT,traits>& operator»(std::basic_istream<charT,traits> &in, half &arg)
    {
    #ifdef HALF_ARITHMETIC_TYPE
        detail::internal_t f;
    #else
        double f;
    #endif
        if(in » f)
            arg.data_ = detail::float2half<half::round_style>(f);
        return in;
    }


    inline HALF_CONSTEXPR half fabs(half arg) { return half(detail::binary, arg.data_&0x7FFF); }

    inline HALF_CONSTEXPR half abs(half arg) { return fabs(arg); }

    inline half fmod(half x, half y)
    {
        unsigned int absx = x.data_ & 0x7FFF, absy = y.data_ & 0x7FFF, sign = x.data_ & 0x8000;
        if(absx >= 0x7C00 || absy >= 0x7C00)
            return half(detail::binary,   (absx>0x7C00 || absy>0x7C00) ? detail::signal(x.data_, y.data_) :
                                        (absx==0x7C00) ? detail::invalid() : x.data_);
        if(!absy)
            return half(detail::binary, detail::invalid());
        if(!absx)
            return x;
        if(absx == absy)
            return half(detail::binary, sign);
        return half(detail::binary, sign|detail::mod<false,false>(absx, absy));
    }

    inline half remainder(half x, half y)
    {
        unsigned int absx = x.data_ & 0x7FFF, absy = y.data_ & 0x7FFF, sign = x.data_ & 0x8000;
        if(absx >= 0x7C00 || absy >= 0x7C00)
            return half(detail::binary,   (absx>0x7C00 || absy>0x7C00) ? detail::signal(x.data_, y.data_) :
                                        (absx==0x7C00) ? detail::invalid() : x.data_);
        if(!absy)
            return half(detail::binary, detail::invalid());
        if(absx == absy)
            return half(detail::binary, sign);
        return half(detail::binary, sign^detail::mod<false,true>(absx, absy));
    }

    inline half remquo(half x, half y, int *quo)
    {
        unsigned int absx = x.data_ & 0x7FFF, absy = y.data_ & 0x7FFF, value = x.data_ & 0x8000;
        if(absx >= 0x7C00 || absy >= 0x7C00)
            return half(detail::binary,   (absx>0x7C00 || absy>0x7C00) ? detail::signal(x.data_, y.data_) :
                                        (absx==0x7C00) ? detail::invalid() : (*quo = 0, x.data_));
        if(!absy)
            return half(detail::binary, detail::invalid());
        bool qsign = ((value^y.data_)&0x8000) != 0;
        int q = 1;
        if(absx != absy)
            value ^= detail::mod<true, true>(absx, absy, &q);
        return *quo = qsign ? -q : q, half(detail::binary, value);
    }

    inline half fma(half x, half y, half z)
    {
    #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_);
        #if HALF_ENABLE_CPP11_CMATH && FP_FAST_FMA
            return half(detail::binary, detail::float2half<half::round_style>(std::fma(fx, fy, fz)));
        #else
            return half(detail::binary, detail::float2half<half::round_style>(fx*fy+fz));
        #endif
    #else
        int absx = x.data_ & 0x7FFF, absy = y.data_ & 0x7FFF, absz = z.data_ & 0x7FFF, exp = -15;
        unsigned int sign = (x.data_^y.data_) & 0x8000;
        bool sub = ((sign^z.data_)&0x8000) != 0;
        if(absx >= 0x7C00 || absy >= 0x7C00 || absz >= 0x7C00)
            return  (absx>0x7C00 || absy>0x7C00 || absz>0x7C00) ? half(detail::binary, detail::signal(x.data_, y.data_, z.data_)) :
                    (absx==0x7C00) ? half(detail::binary, (!absy || (sub && absz==0x7C00)) ? detail::invalid() : (sign|0x7C00)) :
                    (absy==0x7C00) ? half(detail::binary, (!absx || (sub && absz==0x7C00)) ? detail::invalid() : (sign|0x7C00)) : z;
        if(!absx || !absy)
            return absz ? z : half(detail::binary, (half::round_style==std::round_toward_neg_infinity) ? (z.data_|sign) : (z.data_&sign));
        for(; absx<0x400; absx«=1,--exp) ;
        for(; absy<0x400; absy«=1,--exp) ;
        detail::uint32 m = static_cast<detail::uint32>((absx&0x3FF)|0x400) * static_cast<detail::uint32>((absy&0x3FF)|0x400);
        int i = m » 21;
        exp += (absx»10) + (absy»10) + i;
        m «= 3 - i;
        if(absz)
        {
            int expz = 0;
            for(; absz<0x400; absz«=1,--expz) ;
            expz += absz » 10;
            detail::uint32 mz = static_cast<detail::uint32>((absz&0x3FF)|0x400) « 13;
            if(expz > exp || (expz == exp && mz > m))
            {
                std::swap(m, mz);
                std::swap(exp, expz);
                if(sub)
                    sign = z.data_ & 0x8000;
            }
            int d = exp - expz;
            mz = (d<23) ? ((mz»d)|((mz&((static_cast<detail::uint32>(1)«d)-1))!=0)) : 1;
            if(sub)
            {
                m = m - mz;
                if(!m)
                    return half(detail::binary, static_cast<unsigned>(half::round_style==std::round_toward_neg_infinity)«15);
                for(; m<0x800000; m«=1,--exp) ;
            }
            else
            {
                m += mz;
                i = m » 24;
                m = (m»i) | (m&i);
                exp += i;
            }
        }
        if(exp > 30)
            return half(detail::binary, detail::overflow<half::round_style>(sign));
        else if(exp < -10)
            return half(detail::binary, detail::underflow<half::round_style>(sign));
        return half(detail::binary, detail::fixed2half<half::round_style,23,false,false,false>(m, exp-1, sign));
    #endif
    }

    inline HALF_CONSTEXPR_NOERR half fmax(half x, half y)
    {
        return half(detail::binary, (!isnan(y) && (isnan(x) || (x.data_^(0x8000|(0x8000-(x.data_»15)))) <
            (y.data_^(0x8000|(0x8000-(y.data_»15)))))) ? detail::select(y.data_, x.data_) : detail::select(x.data_, y.data_));
    }

    inline HALF_CONSTEXPR_NOERR half fmin(half x, half y)
    {
        return half(detail::binary, (!isnan(y) && (isnan(x) || (x.data_^(0x8000|(0x8000-(x.data_»15)))) >
            (y.data_^(0x8000|(0x8000-(y.data_»15)))))) ? detail::select(y.data_, x.data_) : detail::select(x.data_, y.data_));
    }

    inline half fdim(half x, half y)
    {
        if(isnan(x) || isnan(y))
            return half(detail::binary, detail::signal(x.data_, y.data_));
        return (x.data_^(0x8000|(0x8000-(x.data_»15)))) <= (y.data_^(0x8000|(0x8000-(y.data_»15)))) ?
```

```
half(detail::binary, 0) : (x-y);
    }

    inline half nanh(const char *arg)
    {
        unsigned int value = 0x7FFF;
        while(*arg)
            value ^= static_cast<unsigned>(*arg++) & 0xFF;
        return half(detail::binary, value);
    }


    inline half exp(half arg)
    {
    #ifdef HALF_ARITHMETIC_TYPE
        return half(detail::binary,
detail::float2half<half::round_style>(std::exp(detail::half2float<detail::internal_t>(arg.data_))));
    #else
        int abs = arg.data_ & 0x7FFF, e = (abs»10) + (abs<=0x3FF), exp;
        if(!abs)
            return half(detail::binary, 0x3C00);
        if(abs >= 0x7C00)
            return half(detail::binary, (abs==0x7C00) ? (0x7C00&((arg.data_»15)-1U)) :
detail::signal(arg.data_));
        if(abs >= 0x4C80)
            return half(detail::binary, (arg.data_&0x8000) ? detail::underflow<half::round_style>() :
detail::overflow<half::round_style>());
        detail::uint32 m =
detail::multiply64(static_cast<detail::uint32>((abs&0x3FF)+((abs>0x3FF)«10))«21, 0xB8AA3B29);
        if(e < 14)
        {
            exp = 0;
            m »= 14 - e;
        }
        else
        {
            exp = m » (45-e);
            m = (m«(e-14)) & 0x7FFFFFFF;
        }
        return half(detail::binary, detail::exp2_post<half::round_style>(m, exp,
(arg.data_&0x8000)!=0, 0, 26));
    #endif
    }

    inline half exp2(half arg)
    {
    #if defined(HALF_ARITHMETIC_TYPE) && HALF_ENABLE_CPP11_CMATH
        return half(detail::binary,
detail::float2half<half::round_style>(std::exp2(detail::half2float<detail::internal_t>(arg.data_))));
    #else
        int abs = arg.data_ & 0x7FFF, e = (abs»10) + (abs<=0x3FF), exp = (abs&0x3FF) +
((abs>0x3FF)«10);
        if(!abs)
            return half(detail::binary, 0x3C00);
        if(abs >= 0x7C00)
            return half(detail::binary, (abs==0x7C00) ? (0x7C00&((arg.data_»15)-1U)) :
detail::signal(arg.data_));
        if(abs >= 0x4E40)
            return half(detail::binary, (arg.data_&0x8000) ? detail::underflow<half::round_style>() :
detail::overflow<half::round_style>());
        return half(detail::binary, detail::exp2_post<half::round_style>(
            (static_cast<detail::uint32>(exp)«(6+e))&0x7FFFFFFF, exp»(25-e), (arg.data_&0x8000)!=0, 0,
28));
    #endif
    }

    inline half expm1(half arg)
    {
    #if defined(HALF_ARITHMETIC_TYPE) && HALF_ENABLE_CPP11_CMATH
        return half(detail::binary,
detail::float2half<half::round_style>(std::expm1(detail::half2float<detail::internal_t>(arg.data_))));
    #else
        unsigned int abs = arg.data_ & 0x7FFF, sign = arg.data_ & 0x8000, e = (abs»10) + (abs<=0x3FF),
exp;
        if(!abs)
            return arg;
        if(abs >= 0x7C00)
            return half(detail::binary, (abs==0x7C00) ? (0x7C00+(sign»1)) :
detail::signal(arg.data_));
        if(abs >= 0x4A00)
            return half(detail::binary, (arg.data_&0x8000) ?
detail::rounded<half::round_style,true>(0xBBFF, 1, 1) : detail::overflow<half::round_style>());
        detail::uint32 m =
detail::multiply64(static_cast<detail::uint32>((abs&0x3FF)+((abs>0x3FF)«10))«21, 0xB8AA3B29);
        if(e < 14)
        {
            exp = 0;
            m »= 14 - e;
        }
        else
        {
            exp = m » (45-e);
            m = (m«(e-14)) & 0x7FFFFFFF;
        }
        m = detail::exp2(m);
        if(sign)
        {
            int s = 0;
            if(m > 0x80000000)
            {
                ++exp;
                m = detail::divide64(0x80000000, m, s);
            }
            m = 0x80000000 - ((m»exp)|((m&((static_cast<detail::uint32>(1)«exp)-1))!=0)|s);
            exp = 0;
        }
        else
            m -= (exp<31) ? (0x80000000»exp) : 1;
        for(exp+=14; m<0x80000000 && exp; m«=1,--exp) ;
        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), (m»20)&1, (m&0xFFFFF)!=0));
    #endif
    }

    inline half log(half arg)
    {
    #ifdef HALF_ARITHMETIC_TYPE
        return half(detail::binary, detail::float2half<half::round_style>(std::log(detail::half2float<detail::internal_t>(arg.data_))));
    #else
        int abs = arg.data_ & 0x7FFF, exp = -15;
        if(!abs)
            return half(detail::binary, detail::pole(0x8000));
        if(arg.data_ & 0x8000)
            return half(detail::binary, (arg.data_<=0xFC00) ? detail::invalid() : detail::signal(arg.data_));
        if(abs >= 0x7C00)
            return (abs==0x7C00) ? arg : half(detail::binary, detail::signal(arg.data_));
        for(; abs<0x400; abs«=1,--exp) ;
        exp += abs » 10;
        return half(detail::binary, detail::log2_post<half::round_style,0xB8AA3B2A>(
            detail::log2(static_cast<detail::uint32>((abs&0x3FF)|0x400)«20, 27)+8, exp, 17));
    #endif
    }

    inline half log10(half arg)
    {
    #ifdef HALF_ARITHMETIC_TYPE
        return half(detail::binary, detail::float2half<half::round_style>(std::log10(detail::half2float<detail::internal_t>(arg.data_))));
    #else
        int abs = arg.data_ & 0x7FFF, exp = -15;
        if(!abs)
            return half(detail::binary, detail::pole(0x8000));
        if(arg.data_ & 0x8000)
            return half(detail::binary, (arg.data_<=0xFC00) ? detail::invalid() : detail::signal(arg.data_));
        if(abs >= 0x7C00)
            return (abs==0x7C00) ? arg : half(detail::binary, detail::signal(arg.data_));
        switch(abs)
        {
            case 0x4900: return half(detail::binary, 0x3C00);
            case 0x5640: return half(detail::binary, 0x4000);
            case 0x63D0: return half(detail::binary, 0x4200);
            case 0x70E2: return half(detail::binary, 0x4400);
        }
        for(; abs<0x400; abs«=1,--exp) ;
        exp += abs » 10;
        return half(detail::binary, detail::log2_post<half::round_style,0xD49A784C>(
            detail::log2(static_cast<detail::uint32>((abs&0x3FF)|0x400)«20, 27)+8, exp, 16));
    #endif
    }

    inline half log2(half arg)
    {
    #if defined(HALF_ARITHMETIC_TYPE) && HALF_ENABLE_CPP11_CMATH
        return half(detail::binary, detail::float2half<half::round_style>(std::log2(detail::half2float<detail::internal_t>(arg.data_))));
    #else
        int abs = arg.data_ & 0x7FFF, exp = -15, s = 0;
        if(!abs)
            return half(detail::binary, detail::pole(0x8000));
        if(arg.data_ & 0x8000)
            return half(detail::binary, (arg.data_<=0xFC00) ? detail::invalid() : detail::signal(arg.data_));
        if(abs >= 0x7C00)
            return (abs==0x7C00) ? arg : half(detail::binary, detail::signal(arg.data_));
        if(abs == 0x3C00)
            return half(detail::binary, 0);
        for(; abs<0x400; abs«=1,--exp) ;
        exp += (abs»10);
        if(!(abs&0x3FF))
        {
            unsigned int value = static_cast<unsigned>(exp<0) « 15, m = std::abs(exp) « 6;
            for(exp=18; m<0x400; m«=1,--exp) ;
            return half(detail::binary, value+(exp«10)+m);
        }
        detail::uint32 ilog = exp, sign = detail::sign_mask(ilog), m =
            (((ilog«27)+(detail::log2(static_cast<detail::uint32>((abs&0x3FF)|0x400)«20, 28)»4))^sign) - sign;
        if(!m)
            return half(detail::binary, 0);
        for(exp=14; m<0x8000000 && exp; m«=1,--exp) ;
        for(; m>0xFFFFFFF; m»=1,++exp)
            s |= m & 1;
        return half(detail::binary, detail::fixed2half<half::round_style,27,false,false,true>(m, exp, sign&0x8000, s));
    #endif
    }

    inline half log1p(half arg)
    {
    #if defined(HALF_ARITHMETIC_TYPE) && HALF_ENABLE_CPP11_CMATH
        return half(detail::binary, detail::float2half<half::round_style>(std::log1p(detail::half2float<detail::internal_t>(arg.data_))));
    #else
        if(arg.data_ >= 0xBC00)
            return half(detail::binary, (arg.data_==0xBC00) ? detail::pole(0x8000) : (arg.data_<=0xFC00) ? detail::invalid() : detail::signal(arg.data_));
        int abs = arg.data_ & 0x7FFF, exp = -15;
        if(!abs || abs >= 0x7C00)
            return (abs>0x7C00) ? half(detail::binary, detail::signal(arg.data_)) : arg;
        for(; abs<0x400; abs«=1,--exp) ;
        exp += abs » 10;
        detail::uint32 m = static_cast<detail::uint32>((abs&0x3FF)|0x400) « 20;
        if(arg.data_ & 0x8000)
        {
            m = 0x40000000 - (m»-exp);
            for(exp=0; m<0x40000000; m«=1,--exp) ;
        }
        else
        {
            if(exp < 0)
            {
                m = 0x40000000 + (m»-exp);
                exp = 0;
            }
            else
            {
                m += 0x40000000 » exp;
                int i = m » 31;
                m »= i;
                exp += i;
            }
        }
        return half(detail::binary, detail::log2_post<half::round_style,0xB8AA3B2A>(detail::log2(m), exp, 17));
    #endif
    }



    inline half sqrt(half arg)
    {
    #ifdef HALF_ARITHMETIC_TYPE
        return half(detail::binary, detail::float2half<half::round_style>(std::sqrt(detail::half2float<detail::internal_t>(arg.data_))));
    #else
        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) ? detail::invalid() : arg.data_);
        for(; abs<0x400; abs«=1,--exp) ;
        detail::uint32 r = static_cast<detail::uint32>((abs&0x3FF)|0x400) « 10, m = 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));
    #endif
    }

    inline half rsqrt(half arg)
    {
    #ifdef HALF_ARITHMETIC_TYPE
        return half(detail::binary, detail::float2half<half::round_style>(detail::internal_t(1)/std::sqrt(detail::half2float<detail::internal_t>(arg.data_)
    #else
        unsigned int abs = arg.data_ & 0x7FFF, bias = 0x4000;
        if(!abs || arg.data_ >= 0x7C00)
            return half(detail::binary,   (abs>0x7C00) ? detail::signal(arg.data_) : (arg.data_>0x8000) ?
                                               detail::invalid() : !abs ? detail::pole(arg.data_&0x8000) : 0);
        for(; abs<0x400; abs«=1,bias-=0x400) ;
        unsigned int frac = (abs+=bias) & 0x7FF;
        if(frac == 0x400)
            return half(detail::binary, 0x7A00-(abs»1));
        if((half::round_style == std::round_to_nearest && (frac == 0x3FE || frac == 0x76C)) ||
           (half::round_style != std::round_to_nearest && (frac == 0x15A || frac == 0x3FC || frac == 0x401 || frac == 0x402 || frac == 0x67B)))
            return pow(arg, half(detail::binary, 0xB800));
        detail::uint32 f = 0x17376 - abs, mx = (abs&0x3FF) | 0x400, my = ((f»1)&0x3FF) | 0x400, mz = my * my;
        int expy = (f»11) - 31, expx = 32 - (abs»10), i = mz » 21;
        for(mz=0x60000000-(((mz»i)*mx)»(expx-2*expy-i)); mz<0x40000000; mz«=1,--expy) ;
        i = (my*=mz»10) » 31;
        expy += i;
        my = (my»(20+i)) + 1;
        i = (mz=my*my) » 21;
        for(mz=0x60000000-(((mz»i)*mx)»(expx-2*expy-i)); mz<0x40000000; mz«=1,--expy) ;
        i = (my*=(mz»10)+1) » 31;
        return half(detail::binary, detail::fixed2half<half::round_style,30,false,false,true>(my»i, expy+i+14));
    #endif
    }

    inline half cbrt(half arg)
    {
    #if defined(HALF_ARITHMETIC_TYPE) && HALF_ENABLE_CPP11_CMATH
        return half(detail::binary, detail::float2half<half::round_style>(std::cbrt(detail::half2float<detail::internal_t>(arg.data_))));
    #else
        int abs = arg.data_ & 0x7FFF, exp = -15;
        if(!abs || abs == 0x3C00 || abs >= 0x7C00)
            return (abs>0x7C00) ? half(detail::binary, detail::signal(arg.data_)) : arg;
        for(; abs<0x400; abs«=1, --exp);
        detail::uint32 ilog = exp + (abs»10), sign = detail::sign_mask(ilog), f, m =
            (((ilog«27)+(detail::log2(static_cast<detail::uint32>((abs&0x3FF)|0x400)«20, 24)»4))^sign) - sign;
        for(exp=2; m<0x80000000; m«=1,--exp) ;
        m = detail::multiply64(m, 0xAAAAAAAB);
        int i = m » 31, s;
        exp += i;
        m «= 1 - i;
        if(exp < 0)
        {
            f = m » -exp;
            exp = 0;
        }
        else
        {
            f = (m«exp) & 0x7FFFFFFF;
            exp = m » (31-exp);
        }
        m = detail::exp2(f, (half::round_style==std::round_to_nearest) ? 29 : 26);
        if(sign)
        {
            if(m > 0x80000000)
            {
                m = detail::divide64(0x80000000, m, s);
                ++exp;
            }
            exp = -exp;
        }
        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, arg.data_&0x8000));
    #endif
    }

    inline half hypot(half x, half y)
    {
    #ifdef HALF_ARITHMETIC_TYPE
        detail::internal_t fx = detail::half2float<detail::internal_t>(x.data_), fy = detail::half2float<detail::internal_t>(y.data_);
        #if HALF_ENABLE_CPP11_CMATH
            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)));
        #endif
    #else
        int absx = x.data_ & 0x7FFF, absy = y.data_ & 0x7FFF, expx = 0, expy = 0;
        if(absx >= 0x7C00 || absy >= 0x7C00)
            return half(detail::binary,   (absx==0x7C00) ? detail::select(0x7C00, y.data_) :
                (absy==0x7C00) ? detail::select(0x7C00, x.data_) : detail::signal(x.data_, y.data_));
        if(!absx)
            return half(detail::binary, absy ? detail::check_underflow(absy) : 0);
        if(!absy)
            return half(detail::binary, detail::check_underflow(absx));
        if(absy > absx)
            std::swap(absx, absy);
        for(; absx<0x400; absx«=1,--expx) ;
        for(; absy<0x400; absy«=1,--expy) ;
        detail::uint32 mx = (absx&0x3FF) | 0x400, my = (absy&0x3FF) | 0x400;
        mx *= mx;
        my *= my;
        int ix = mx » 21, iy = my » 21;
        expx = 2*(expx+(absx»10)) - 15 + ix;
        expy = 2*(expy+(absy»10)) - 15 + iy;
        mx «= 10 - ix;
        my «= 10 - iy;
        int d = expx - expy;
        my = (d<30) ? ((my»d)|((my&((static_cast<detail::uint32>(1)«d)-1))!=0)) : 1;
        return half(detail::binary, detail::hypot_post<half::round_style>(mx+my, expx));
    #endif
    }

    inline half hypot(half x, half y, half z)
    {
    #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_);
        return half(detail::binary, detail::float2half<half::round_style>(std::sqrt(fx*fx+fy*fy+fz*fz)));
    #else
        int absx = x.data_ & 0x7FFF, absy = y.data_ & 0x7FFF, absz = z.data_ & 0x7FFF, expx = 0, expy = 0, expz = 0;
        if(!absx)
            return hypot(y, z);
        if(!absy)
            return hypot(x, z);
        if(!absz)
            return hypot(x, y);
        if(absx >= 0x7C00 || absy >= 0x7C00 || absz >= 0x7C00)
            return half(detail::binary,   (absx==0x7C00) ? detail::select(0x7C00, detail::select(y.data_, z.data_)) :
                                        (absy==0x7C00) ? detail::select(0x7C00, detail::select(x.data_, z.data_)) :
                                        (absz==0x7C00) ? detail::select(0x7C00, detail::select(x.data_, y.data_)) :
                                        detail::signal(x.data_, y.data_, z.data_));
        if(absz > absy)
            std::swap(absy, absz);
        if(absy > absx)
            std::swap(absx, absy);
        if(absz > absy)
            std::swap(absy, absz);
        for(; absx<0x400; absx«=1,--expx) ;
        for(; absy<0x400; absy«=1,--expy) ;
        for(; absz<0x400; absz«=1,--expz) ;
        detail::uint32 mx = (absx&0x3FF) | 0x400, my = (absy&0x3FF) | 0x400, mz = (absz&0x3FF) | 0x400;
        mx *= mx;
        my *= my;
        mz *= mz;
        int ix = mx » 21, iy = my » 21, iz = mz » 21;
        expx = 2*(expx+(absx»10)) - 15 + ix;
        expy = 2*(expy+(absy»10)) - 15 + iy;
        expz = 2*(expz+(absz»10)) - 15 + iz;
        mx «= 10 - ix;
        my «= 10 - iy;
        mz «= 10 - iz;
        int d = expy - expz;
        mz = (d<30) ? ((mz»d)|((mz&((static_cast<detail::uint32>(1)«d)-1))!=0)) : 1;
        my += mz;
        if(my & 0x80000000)
        {
            my = (my»1) | (my&1);
            if(++expy > expx)
            {
                std::swap(mx, my);
                std::swap(expx, expy);
            }
        }
        d = expx - expy;
        my = (d<30) ? ((my»d)|((my&((static_cast<detail::uint32>(1)«d)-1))!=0)) : 1;
        return half(detail::binary, detail::hypot_post<half::round_style>(mx+my, expx));
    #endif
    }

    inline half pow(half x, half y)
    {
    #ifdef HALF_ARITHMETIC_TYPE
        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_))));
    #else
        int absx = x.data_ & 0x7FFF, absy = y.data_ & 0x7FFF, exp = -15;
        if(!absy || x.data_ == 0x3C00)
            return half(detail::binary, detail::select(0x3C00, (x.data_==0x3C00) ? y.data_ :
x.data_));
        bool is_int = absy >= 0x6400 || (absy>=0x3C00 && !(absy&((1«(25-(absy»10)))-1)));
        unsigned int sign = x.data_ &
(static_cast<unsigned>((absy<0x6800)&&is_int&&((absy»(25-(absy»10)))&1))«15);
        if(absx >= 0x7C00 || absy >= 0x7C00)
            return half(detail::binary,   (absx>0x7C00 || absy>0x7C00) ? detail::signal(x.data_,
y.data_) :
                                         (absy==0x7C00) ? ((absx==0x3C00) ? 0x3C00 : (!absx &&
y.data_==0xFC00) ? detail::pole() :
                                         (0x7C00&-((y.data_»15)^(absx>0x3C00)))) :
(sign|(0x7C00&((y.data_»15)-1U))));
        if(!absx)
            return half(detail::binary, (y.data_&0x8000) ? detail::pole(sign) : sign);
        if((x.data_&0x8000) && !is_int)
            return half(detail::binary, detail::invalid());
        if(x.data_ == 0xBC00)
            return half(detail::binary, sign|0x3C00);
        switch(y.data_)
        {
            case 0x3800: return sqrt(x);
            case 0x3C00: return half(detail::binary, detail::check_underflow(x.data_));
            case 0x4000: return x * x;
            case 0xBC00: return half(detail::binary, 0x3C00) / x;
        }
        for(; absx<0x400; absx«=1,--exp) ;
        detail::uint32 ilog = exp + (absx»10), msign = detail::sign_mask(ilog), f, m =

(((ilog«27)+((detail::log2(static_cast<detail::uint32>((absx&0x3FF)|0x400)«20)+8)»4))^msign) - msign;
        for(exp=-11; m<0x80000000; m«=1,--exp) ;
        for(; absy<0x400; absy«=1,--exp) ;
        m = detail::multiply64(m, static_cast<detail::uint32>((absy&0x3FF)|0x400)«21);
        int i = m » 31;
        exp += (absy»10) + i;
        m «= 1 - i;
        if(exp < 0)
        {
            f = m » -exp;
            exp = 0;
        }
        else
        {
            f = (m«exp) & 0x7FFFFFFF;
            exp = m » (31-exp);
        }
        return half(detail::binary, detail::exp2_post<half::round_style>(f, exp,
((msign&1)^(y.data_»15))!=0, sign));
    #endif
    }


    inline void sincos(half arg, half *sin, half *cos)
    {
    #ifdef HALF_ARITHMETIC_TYPE
        detail::internal_t f = detail::half2float<detail::internal_t>(arg.data_);
        *sin = half(detail::binary, detail::float2half<half::round_style>(std::sin(f)));
        *cos = half(detail::binary, detail::float2half<half::round_style>(std::cos(f)));
    #else
        int abs = arg.data_ & 0x7FFF, sign = arg.data_ » 15, k;
        if(abs >= 0x7C00)
            *sin = *cos = half(detail::binary, (abs==0x7C00) ? detail::invalid() :
detail::signal(arg.data_));
        else if(!abs)
        {
            *sin = arg;
            *cos = half(detail::binary, 0x3C00);
        }
        else if(abs < 0x2500)
        {
            *sin = half(detail::binary, detail::rounded<half::round_style,true>(arg.data_-1, 1, 1));
            *cos = half(detail::binary, detail::rounded<half::round_style,true>(0x3BFF, 1, 1));
        }
        else
        {
            if(half::round_style != std::round_to_nearest)
            {
                switch(abs)
                {
                case 0x48B7:
                    *sin = half(detail::binary, detail::rounded<half::round_style,true>((~arg.data_&0x8000)|0x1D07, 1, 1));
                    *cos = half(detail::binary, detail::rounded<half::round_style,true>(0xBBFF, 1, 1));
                    return;
                case 0x598C:
                    *sin = half(detail::binary, detail::rounded<half::round_style,true>((arg.data_&0x8000)|0x3BFF, 1, 1));
                    *cos = half(detail::binary, detail::rounded<half::round_style,true>(0x80FC, 1, 1));
                    return;
                case 0x6A64:
                    *sin = half(detail::binary, detail::rounded<half::round_style,true>((~arg.data_&0x8000)|0x3BFE, 1, 1));
                    *cos = half(detail::binary, detail::rounded<half::round_style,true>(0x27FF, 1, 1));
                    return;
                case 0x6D8C:
                    *sin = half(detail::binary, detail::rounded<half::round_style,true>((arg.data_&0x8000)|0x0FE6, 1, 1));
                    *cos = half(detail::binary, detail::rounded<half::round_style,true>(0x3BFF, 1, 1));
                    return;
                }
            }
            std::pair<detail::uint32,detail::uint32> sc = detail::sincos(detail::angle_arg(abs, k), 28);
            switch(k & 3)
            {
                case 1: sc = std::make_pair(sc.second, -sc.first); break;
                case 2: sc = std::make_pair(-sc.first, -sc.second); break;
                case 3: sc = std::make_pair(-sc.second, sc.first); break;
            }
            *sin = half(detail::binary, detail::fixed2half<half::round_style,30,true,true,true>((sc.first^-static_cast<detail::uint32>(sign))+sign));
            *cos = half(detail::binary, detail::fixed2half<half::round_style,30,true,true,true>(sc.second));
        }
    #endif
    }

    inline half sin(half arg)
    {
    #ifdef HALF_ARITHMETIC_TYPE
        return half(detail::binary, detail::float2half<half::round_style>(std::sin(detail::half2float<detail::internal_t>(arg.data_))));
    #else
        int abs = arg.data_ & 0x7FFF, k;
        if(!abs)
            return arg;
        if(abs >= 0x7C00)
            return half(detail::binary, (abs==0x7C00) ? detail::invalid() : detail::signal(arg.data_));
        if(abs < 0x2900)
            return half(detail::binary, detail::rounded<half::round_style,true>(arg.data_-1, 1, 1));
        if(half::round_style != std::round_to_nearest)
            switch(abs)
            {
                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));
                case 0x6D8C: return half(detail::binary, detail::rounded<half::round_style,true>((arg.data_&0x8000)|0x0FE6, 1, 1));
            }
        std::pair<detail::uint32,detail::uint32> sc = detail::sincos(detail::angle_arg(abs, k), 28);
        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));
    #endif
    }

    inline half cos(half arg)
    {
    #ifdef HALF_ARITHMETIC_TYPE
        return half(detail::binary, detail::float2half<half::round_style>(std::cos(detail::half2float<detail::internal_t>(arg.data_))));
    #else
        int abs = arg.data_ & 0x7FFF, k;
        if(!abs)
            return half(detail::binary, 0x3C00);
        if(abs >= 0x7C00)
            return half(detail::binary, (abs==0x7C00) ? detail::invalid() : detail::signal(arg.data_));
        if(abs < 0x2500)
            return half(detail::binary, detail::rounded<half::round_style,true>(0x3BFF, 1, 1));
        if(half::round_style != std::round_to_nearest && abs == 0x598C)
            return half(detail::binary, detail::rounded<half::round_style,true>(0x80FC, 1, 1));
        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);
        return half(detail::binary, detail::fixed2half<half::round_style,30,true,true,true>((((k&1) ? sc.first : sc.second)^sign) - sign));
    #endif
    }

    inline half tan(half arg)
    {
    #ifdef HALF_ARITHMETIC_TYPE
        return half(detail::binary, detail::float2half<half::round_style>(std::tan(detail::half2float<detail::internal_t>(arg.data_))));
    #else
        int abs = arg.data_ & 0x7FFF, exp = 13, k;
        if(!abs)
            return arg;
        if(abs >= 0x7C00)
            return half(detail::binary, (abs==0x7C00) ? detail::invalid() : detail::signal(arg.data_));
        if(abs < 0x2700)
            return half(detail::binary, detail::rounded<half::round_style,true>(arg.data_, 0, 1));
        if(half::round_style != std::round_to_nearest)
            switch(abs)
            {
                case 0x658C: return half(detail::binary, detail::rounded<half::round_style,true>((arg.data_&0x8000)|0x07E6, 1, 1));
                case 0x7330: return half(detail::binary, detail::rounded<half::round_style,true>((~arg.data_&0x8000)|0x4B62, 1, 1));
            }
        std::pair<detail::uint32,detail::uint32> sc = detail::sincos(detail::angle_arg(abs, k), 30);
        if(k & 1)
            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;
        for(; my<0x80000000; my«=1,--exp) ;
        for(; mx<0x80000000; mx«=1,++exp) ;
        return half(detail::binary, detail::tangent_post<half::round_style>(my, mx, exp, (signy^signx^arg.data_)&0x8000));
    #endif
    }

    inline half asin(half arg)
    {
    #ifdef HALF_ARITHMETIC_TYPE
        return half(detail::binary, detail::float2half<half::round_style>(std::asin(detail::half2float<detail::internal_t>(arg.data_))));
    #else
        unsigned int abs = arg.data_ & 0x7FFF, sign = arg.data_ & 0x8000;
        if(!abs)
            return arg;
        if(abs >= 0x3C00)
            return half(detail::binary, (abs>0x7C00) ? detail::signal(arg.data_) : (abs>0x3C00) ? detail::invalid() :
                                        detail::rounded<half::round_style,true>(sign|0x3E48, 0, 1));
        if(abs < 0x2900)
            return half(detail::binary, detail::rounded<half::round_style,true>(arg.data_, 0, 1));
        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);
        detail::uint32 m = detail::atan2(sc.first, sc.second, (half::round_style==std::round_to_nearest) ? 27 : 26);
        return half(detail::binary, detail::fixed2half<half::round_style,30,false,true,true>(m, 14, sign));
    #endif
    }

    inline half acos(half arg)
    {
    #ifdef HALF_ARITHMETIC_TYPE
        return half(detail::binary, detail::float2half<half::round_style>(std::acos(detail::half2float<detail::internal_t>(arg.data_))));
    #else
        unsigned int abs = arg.data_ & 0x7FFF, sign = arg.data_ » 15;
        if(!abs)
            return half(detail::binary, detail::rounded<half::round_style,true>(0x3E48, 0, 1));
        if(abs >= 0x3C00)
            return half(detail::binary,   (abs>0x7C00) ? detail::signal(arg.data_) : (abs>0x3C00) ? detail::invalid() :
```

```
                                        sign ? detail::rounded<half::round_style,true>(0x4248, 0, 1) : 0);
        std::pair<detail::uint32,detail::uint32> cs = detail::atan2_args(abs);
        detail::uint32 m = detail::atan2(cs.second, cs.first, 28);
        return half(detail::binary, detail::fixed2half<half::round_style,31,false,true,true>(sign ? (0xC90FDAA2-m) : m, 15, 0, sign));
    #endif
    }

    inline half atan(half arg)
    {
    #ifdef HALF_ARITHMETIC_TYPE
        return half(detail::binary, detail::float2half<half::round_style>(std::atan(detail::half2float<detail::internal_t>(arg.data_))));
    #else
        unsigned int abs = arg.data_ & 0x7FFF, sign = arg.data_ & 0x8000;
        if(!abs)
            return arg;
        if(abs >= 0x7C00)
            return half(detail::binary, (abs==0x7C00) ? detail::rounded<half::round_style,true>(sign|0x3E48, 0, 1) : detail::signal(arg.data_));
        if(abs <= 0x2700)
            return half(detail::binary, detail::rounded<half::round_style,true>(arg.data_-1, 1, 1));
        int exp = (abs»10) + (abs<=0x3FF);
        detail::uint32 my = (abs&0x3FF) | ((abs>0x3FF)«10);
        detail::uint32 m = (exp>15) ?  detail::atan2(my«19, 0x20000000»(exp-15), (half::round_style==std::round_to_nearest) ? 26 : 24) :
                                        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, sign));
    #endif
    }

    inline half atan2(half y, half x)
    {
    #ifdef HALF_ARITHMETIC_TYPE
        return half(detail::binary, detail::float2half<half::round_style>(std::atan2(detail::half2float<detail::internal_t>(y.data_), detail::half2float<detail::internal_t>(x.data_))));
    #else
        unsigned int absx = x.data_ & 0x7FFF, absy = y.data_ & 0x7FFF, signx = x.data_ » 15, signy = y.data_ & 0x8000;
        if(absx >= 0x7C00 || absy >= 0x7C00)
        {
            if(absx > 0x7C00 || absy > 0x7C00)
                return half(detail::binary, detail::signal(x.data_, y.data_));
            if(absy == 0x7C00)
                return half(detail::binary, (absx<0x7C00) ? detail::rounded<half::round_style,true>(signy|0x3E48, 0, 1) :
                                                    signx ? detail::rounded<half::round_style,true>(signy|0x40B6, 0, 1) :
                                                    detail::rounded<half::round_style,true>(signy|0x3A48, 0, 1));
            return (x.data_==0x7C00) ? half(detail::binary, signy) : half(detail::binary, detail::rounded<half::round_style,true>(signy|0x4248, 0, 1));
        }
        if(!absy)
            return signx ? half(detail::binary, detail::rounded<half::round_style,true>(signy|0x4248, 0, 1)) : y;
        if(!absx)
            return half(detail::binary, detail::rounded<half::round_style,true>(signy|0x3E48, 0, 1));
        int d = (absy»10) + (absy<=0x3FF) - (absx»10) - (absx<=0x3FF);
        if(d > (signx ? 18 : 12))
            return half(detail::binary, detail::rounded<half::round_style,true>(signy|0x3E48, 0, 1));
        if(signx && d < -11)
            return half(detail::binary, detail::rounded<half::round_style,true>(signy|0x4248, 0, 1));
        if(!signx && d < ((half::round_style==std::round_toward_zero) ? -15 : -9))
        {
            for(; absy<0x400; absy«=1,--d) ;
            detail::uint32 mx = ((absx«1)&0x7FF) | 0x800, my = ((absy«1)&0x7FF) | 0x800;
            int i = my < mx;
            d -= i;
            if(d < -25)
                return half(detail::binary, detail::underflow<half::round_style>(signy));
            my «= 11 + i;
            return half(detail::binary, detail::fixed2half<half::round_style,11,false,false,true>(my/mx, d+14, signy, my%mx!=0));
        }
        detail::uint32 m = detail::atan2(    ((absy&0x3FF)|((absy>0x3FF)«10))«(19+((d<0) ? d : (d>0) ? 0 : -1)),
                                            ((absx&0x3FF)|((absx>0x3FF)«10))«(19-((d>0) ? d : (d<0) ? 0 : 1)));
        return half(detail::binary, detail::fixed2half<half::round_style,31,false,true,true>(signx ? (0xC90FDAA2-m) : m, 15, signy, signx));
    #endif
    }



    inline half sinh(half arg)
    {
    #ifdef HALF_ARITHMETIC_TYPE
        return half(detail::binary, detail::float2half<half::round_style>(std::sinh(detail::half2float<detail::internal_t>(arg.data_))));
    #else
        int abs = arg.data_ & 0x7FFF, exp;
        if(!abs || abs >= 0x7C00)
            return (abs>0x7C00) ? half(detail::binary, detail::signal(arg.data_)) : arg;
        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, (half::round_style==std::round_to_nearest) ? 29 : 27);
        detail::uint32 m = mm.first - mm.second;
        for(exp+=13; m<0x80000000 && exp; m«=1,--exp) ;
        unsigned int sign = arg.data_ & 0x8000;
        if(exp > 29)
            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, sign));
    #endif
    }

    inline half cosh(half arg)
    {
    #ifdef HALF_ARITHMETIC_TYPE
        return half(detail::binary, detail::float2half<half::round_style>(std::cosh(detail::half2float<detail::internal_t>(arg.data_))));
    #else
        int abs = arg.data_ & 0x7FFF, exp;
        if(!abs)
            return half(detail::binary, 0x3C00);
        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, (half::round_style==std::round_to_nearest) ? 23 : 26);
        detail::uint32 m = mm.first + mm.second, i = (~m&0xFFFFFFFF) » 31;
        m = (m»i) | (m&i) | 0x80000000;
        if((exp+=13+i) > 29)
            return half(detail::binary, detail::overflow<half::round_style>());
        return half(detail::binary, detail::fixed2half<half::round_style,31,false,false,true>(m, exp));
    #endif
    }

    inline half tanh(half arg)
    {
    #ifdef HALF_ARITHMETIC_TYPE
        return half(detail::binary, detail::float2half<half::round_style>(std::tanh(detail::half2float<detail::internal_t>(arg.data_))));
    #else
        int abs = arg.data_ & 0x7FFF, exp;
        if(!abs)
            return arg;
        if(abs >= 0x7C00)
            return half(detail::binary, (abs>0x7C00) ? detail::signal(arg.data_) : (arg.data_-0x4000));
        if(abs >= 0x4500)
            return half(detail::binary, detail::rounded<half::round_style,true>((arg.data_&0x8000)|0x3BFF, 1, 1));
        if(abs < 0x2700)
            return half(detail::binary, detail::rounded<half::round_style,true>(arg.data_-1, 1, 1));
        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));
        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;
        for(exp=13; my<0x80000000; my«=1,--exp) ;
        mx = (mx»i) | 0x80000000;
        return half(detail::binary, detail::tangent_post<half::round_style>(my, mx, exp-i, arg.data_&0x8000));
    #endif
    }

    inline half asinh(half arg)
    {
    #if defined(HALF_ARITHMETIC_TYPE) && HALF_ENABLE_CPP11_CMATH
        return half(detail::binary, detail::float2half<half::round_style>(std::asinh(detail::half2float<detail::internal_t>(arg.data_))));
    #else
        int abs = arg.data_ & 0x7FFF;
        if(!abs || abs >= 0x7C00)
            return (abs>0x7C00) ? half(detail::binary, detail::signal(arg.data_)) : arg;
        if(abs <= 0x2900)
            return half(detail::binary, detail::rounded<half::round_style,true>(arg.data_-1, 1, 1));
        if(half::round_style != std::round_to_nearest)
            switch(abs)
            {
                case 0x32D4: return half(detail::binary, detail::rounded<half::round_style,true>(arg.data_-13, 1, 1));
                case 0x3B5B: return half(detail::binary, detail::rounded<half::round_style,true>(arg.data_-197, 1, 1));
            }
        return half(detail::binary, detail::area<half::round_style,true>(arg.data_));
    #endif
    }

    inline half acosh(half arg)
    {
    #if defined(HALF_ARITHMETIC_TYPE) && HALF_ENABLE_CPP11_CMATH
        return half(detail::binary, detail::float2half<half::round_style>(std::acosh(detail::half2float<detail::internal_t>(arg.data_))));
    #else
        int abs = arg.data_ & 0x7FFF;
        if((arg.data_&0x8000) || abs < 0x3C00)
            return half(detail::binary, (abs<=0x7C00) ? detail::invalid() : detail::signal(arg.data_));
        if(abs == 0x3C00)
            return half(detail::binary, 0);
        if(arg.data_ >= 0x7C00)
            return (abs>0x7C00) ? half(detail::binary, detail::signal(arg.data_)) : arg;
        return half(detail::binary, detail::area<half::round_style,false>(arg.data_));
    #endif
    }

    inline half atanh(half arg)
    {
    #if defined(HALF_ARITHMETIC_TYPE) && HALF_ENABLE_CPP11_CMATH
        return half(detail::binary, detail::float2half<half::round_style>(std::atanh(detail::half2float<detail::internal_t>(arg.data_))));
    #else
        int abs = arg.data_ & 0x7FFF, exp = 0;
        if(!abs)
            return arg;
        if(abs >= 0x3C00)
            return half(detail::binary, (abs==0x3C00) ? detail::pole(arg.data_&0x8000) : (abs<=0x7C00) ? detail::invalid() : detail::signal(arg.data_));
        if(abs < 0x2700)
            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)«10)) « ((abs»10)+(abs<=0x3FF)+6), my = 0x80000000 + m, mx = 0x80000000 - m;
        for(; mx<0x80000000; mx«=1,++exp) ;
        int i = my >= mx, s;
        return half(detail::binary, detail::log2_post<half::round_style,0xB8AA3B2A>(detail::log2(
            (detail::divide64(my»i, mx, s)+1)»1, 27)+0x10, exp+i-1, 16, arg.data_&0x8000));
    #endif
    }


    inline half erf(half arg)
    {
    #if defined(HALF_ARITHMETIC_TYPE) && HALF_ENABLE_CPP11_CMATH
        return half(detail::binary, detail::float2half<half::round_style>(std::erf(detail::half2float<detail::internal_t>(arg.data_))));
    #else
        unsigned int abs = arg.data_ & 0x7FFF;
        if(!abs || abs >= 0x7C00)
            return (abs>=0x7C00) ? half(detail::binary, (abs==0x7C00) ? (arg.data_-0x4000) : detail::signal(arg.data_)) : arg;
        if(abs >= 0x4200)
            return half(detail::binary, detail::rounded<half::round_style,true>((arg.data_&0x8000)|0x3BFF, 1, 1));
        return half(detail::binary, detail::erf<half::round_style,false>(arg.data_));
    #endif
    }

    inline half erfc(half arg)
    {
    #if defined(HALF_ARITHMETIC_TYPE) && HALF_ENABLE_CPP11_CMATH
        return half(detail::binary, detail::float2half<half::round_style>(std::erfc(detail::half2float<detail::internal_t>(arg.data_))));
    #else
        unsigned int abs = arg.data_ & 0x7FFF, sign = arg.data_ & 0x8000;
        if(abs >= 0x7C00)
            return (abs>=0x7C00) ? half(detail::binary, (abs==0x7C00) ? (sign»1) : detail::signal(arg.data_)) : arg;
        if(!abs)
            return half(detail::binary, 0x3C00);
        if(abs >= 0x4400)
            return half(detail::binary, detail::rounded<half::round_style,true>((sign»1)-(sign»15), sign»15, 1));
        return half(detail::binary, detail::erf<half::round_style,true>(arg.data_));
    #endif
    }

    inline half lgamma(half arg)
    {
    #if defined(HALF_ARITHMETIC_TYPE) && HALF_ENABLE_CPP11_CMATH
        return half(detail::binary, detail::float2half<half::round_style>(std::lgamma(detail::half2float<detail::internal_t>(arg.data_))));
    #else
        int abs = arg.data_ & 0x7FFF;
        if(abs >= 0x7C00)
            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());
        if(arg.data_ == 0x3C00 || arg.data_ == 0x4000)
            return half(detail::binary, 0);
        return half(detail::binary, detail::gamma<half::round_style,true>(arg.data_));
    #endif
    }

    inline half tgamma(half arg)
    {
    #if defined(HALF_ARITHMETIC_TYPE) && HALF_ENABLE_CPP11_CMATH
        return half(detail::binary, detail::float2half<half::round_style>(std::tgamma(detail::half2float<detail::internal_t>(arg.data_))));
    #else
        unsigned int abs = arg.data_ & 0x7FFF;
        if(!abs)
            return half(detail::binary, detail::pole(arg.data_));
        if(abs >= 0x7C00)
            return (arg.data_==0x7C00) ? arg : half(detail::binary, detail::signal(arg.data_));
        if(arg.data_ >= 0xE400 || (arg.data_ >= 0xBC00 && !(abs&((1«(25-(abs»10)))-1))))
            return half(detail::binary, detail::invalid());
        if(arg.data_ >= 0xCA80)
            return half(detail::binary, detail::underflow<half::round_style>((1-((abs»(25-(abs»10)))&1))«15));
        if(arg.data_ <= 0x100 || (arg.data_ >= 0x4900 && arg.data_ < 0x8000))
            return half(detail::binary, detail::overflow<half::round_style>());
        if(arg.data_ == 0x3C00)
            return arg;
        return half(detail::binary, detail::gamma<half::round_style,false>(arg.data_));
    #endif
    }


    inline half ceil(half arg) { return half(detail::binary, detail::integral<std::round_toward_infinity,true,true>(arg.data_)); }

    inline half floor(half arg) { return half(detail::binary, detail::integral<std::round_toward_neg_infinity,true,true>(arg.data_)); }

    inline half trunc(half arg) { return half(detail::binary, detail::integral<std::round_toward_zero,true,true>(arg.data_)); }

    inline half round(half arg) { return half(detail::binary, detail::integral<std::round_to_nearest,false,true>(arg.data_)); }

    inline long lround(half arg) { return detail::half2int<std::round_to_nearest,false,false,long>(arg.data_); }

    inline half rint(half arg) { return half(detail::binary, detail::integral<half::round_style,true,true>(arg.data_)); }

    inline long lrint(half arg) { return detail::half2int<half::round_style,true,true,long>(arg.data_); }

    inline half nearbyint(half arg) { return half(detail::binary, detail::integral<half::round_style,true,false>(arg.data_)); }
#if HALF_ENABLE_CPP11_LONG_LONG
    inline long long llround(half arg) { return detail::half2int<std::round_to_nearest,false,false,long long>(arg.data_); }

    inline long long llrint(half arg) { return detail::half2int<half::round_style,true,true,long long>(arg.data_); }
#endif


    inline half frexp(half arg, int *exp)
    {
        *exp = 0;
        unsigned int abs = arg.data_ & 0x7FFF;
        if(abs >= 0x7C00 || !abs)
            return (abs>0x7C00) ? half(detail::binary, detail::signal(arg.data_)) : arg;
        for(; abs<0x400; abs«=1,--*exp) ;
        *exp += (abs»10) - 14;
        return half(detail::binary, (arg.data_&0x8000)|0x3800|(abs&0x3FF));
    }

    inline half scalbln(half arg, long exp)
    {
        unsigned int abs = arg.data_ & 0x7FFF, sign = arg.data_ & 0x8000;
        if(abs >= 0x7C00 || !abs)
            return (abs>0x7C00) ? half(detail::binary, detail::signal(arg.data_)) : arg;
        for(; abs<0x400; abs«=1,--exp) ;
        exp += abs » 10;
        if(exp > 30)
            return half(detail::binary, detail::overflow<half::round_style>(sign));
        else if(exp < -10)
            return half(detail::binary, detail::underflow<half::round_style>(sign));
        else if(exp > 0)
            return half(detail::binary, sign|(exp«10)|(abs&0x3FF));
        unsigned int m = (abs&0x3FF) | 0x400;
        return half(detail::binary, detail::rounded<half::round_style,false>(sign|(m»(1-exp)), (m»-exp)&1, (m&((1«-exp)-1))!=0));
    }

    inline half scalbn(half arg, int exp) { return scalbln(arg, exp); }

    inline half ldexp(half arg, int exp) { return scalbln(arg, exp); }

    inline half modf(half arg, half *iptr)
    {
        unsigned int abs = arg.data_ & 0x7FFF;
        if(abs > 0x7C00)
        {
            arg = half(detail::binary, detail::signal(arg.data_));
            return *iptr = arg, arg;
        }
        if(abs >= 0x6400)
            return *iptr = arg, half(detail::binary, arg.data_&0x8000);
        if(abs < 0x3C00)
            return iptr->data_ = arg.data_ & 0x8000, arg;
        unsigned int exp = abs » 10, mask = (1«(25-exp)) - 1, m = arg.data_ & mask;
        iptr->data_ = arg.data_ & ~mask;
        if(!m)
            return half(detail::binary, arg.data_&0x8000);
        for(; m<0x400; m«=1,--exp) ;
        return half(detail::binary, (arg.data_&0x8000)|(exp«10)|(m&0x3FF));
    }

    inline int ilogb(half arg)
    {
        int abs = arg.data_ & 0x7FFF, exp;
        if(!abs || abs >= 0x7C00)
        {
            detail::raise(FE_INVALID);
            return !abs ? FP_ILOGB0 : (abs==0x7C00) ? INT_MAX : FP_ILOGBNAN;
        }
        for(exp=(abs»10)-15; abs<0x200; abs«=1,--exp) ;
        return exp;
    }

    inline half logb(half arg)
    {
        int abs = arg.data_ & 0x7FFF, exp;
        if(!abs)
            return half(detail::binary, detail::pole(0x8000));
        if(abs >= 0x7C00)
            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;
        if(exp)
        {
            unsigned int m = std::abs(exp) « 6;
            for(exp=18; m<0x400; m«=1,--exp) ;
            value |= (exp«10) + m;
        }
        return half(detail::binary, value);
    }

    inline half nextafter(half from, half to)
    {
        int fabs = from.data_ & 0x7FFF, tabs = to.data_ & 0x7FFF;
        if(fabs > 0x7C00 || tabs > 0x7C00)
            return half(detail::binary, detail::signal(from.data_, to.data_));
        if(from.data_ == to.data_ || !(fabs|tabs))
            return to;
        if(!fabs)
        {
            detail::raise(FE_UNDERFLOW, !HALF_ERRHANDLING_UNDERFLOW_TO_INEXACT);
            return half(detail::binary, (to.data_&0x8000)+1);
        }
        unsigned int out = from.data_ + (((from.data_»15)^static_cast<unsigned>(

(from.data_^(0x8000|(0x8000-(from.data_»15))))<(to.data_^(0x8000|(0x8000-(to.data_»15))))))«1) - 1;
        detail::raise(FE_OVERFLOW, fabs<0x7C00 && (out&0x7C00)==0x7C00);
        detail::raise(FE_UNDERFLOW, !HALF_ERRHANDLING_UNDERFLOW_TO_INEXACT && (out&0x7C00)<0x400);
        return half(detail::binary, out);
    }

    inline half nexttoward(half from, long double to)
    {
        int fabs = from.data_ & 0x7FFF;
        if(fabs > 0x7C00)
            return half(detail::binary, detail::signal(from.data_));
        long double lfrom = static_cast<long double>(from);
        if(detail::builtin_isnan(to) || lfrom == to)
            return half(static_cast<float>(to));
        if(!fabs)
        {
            detail::raise(FE_UNDERFLOW, !HALF_ERRHANDLING_UNDERFLOW_TO_INEXACT);
            return half(detail::binary, (static_cast<unsigned>(detail::builtin_signbit(to))«15)+1);
        }
        unsigned int out = from.data_ + (((from.data_»15)^static_cast<unsigned>(lfrom<to))«1) - 1;
        detail::raise(FE_OVERFLOW, (out&0x7FFF)==0x7C00);
        detail::raise(FE_UNDERFLOW, !HALF_ERRHANDLING_UNDERFLOW_TO_INEXACT && (out&0x7FFF)<0x400);
        return half(detail::binary, out);
    }

    inline HALF_CONSTEXPR half copysign(half x, half y) { return half(detail::binary,
x.data_^((x.data_^y.data_)&0x8000)); }



    inline HALF_CONSTEXPR int fpclassify(half arg)
    {
        return  !(arg.data_&0x7FFF) ? FP_ZERO :
                ((arg.data_&0x7FFF)<0x400) ? FP_SUBNORMAL :
                ((arg.data_&0x7FFF)<0x7C00) ? FP_NORMAL :
                ((arg.data_&0x7FFF)==0x7C00) ? FP_INFINITE :
                FP_NAN;
    }

    inline HALF_CONSTEXPR bool isfinite(half arg) { return (arg.data_&0x7C00) != 0x7C00; }

    inline HALF_CONSTEXPR bool isinf(half arg) { return (arg.data_&0x7FFF) == 0x7C00; }

    inline HALF_CONSTEXPR bool isnan(half arg) { return (arg.data_&0x7FFF) > 0x7C00; }

    inline HALF_CONSTEXPR bool isnormal(half arg) { return ((arg.data_&0x7C00)!=0) &
((arg.data_&0x7C00)!=0x7C00); }

    inline HALF_CONSTEXPR bool signbit(half arg) { return (arg.data_&0x8000) != 0; }



    inline HALF_CONSTEXPR bool isgreater(half x, half y)
    {
        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);
    }

    inline HALF_CONSTEXPR bool isgreaterequal(half x, half y)
    {
        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);
    }

    inline HALF_CONSTEXPR bool isless(half x, half y)
    {
        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);
    }

    inline HALF_CONSTEXPR bool islessequal(half x, half y)
    {
        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);
    }

    inline HALF_CONSTEXPR bool islessgreater(half x, half y)
    {
        return x.data_!=y.data_ && ((x.data_|y.data_)&0x7FFF) && !isnan(x) && !isnan(y);
    }

    inline HALF_CONSTEXPR bool isunordered(half x, half y) { return isnan(x) || isnan(y); }



    template<typename T,typename U> T half_cast(U arg) { return detail::half_caster<T,U>::cast(arg); }

    template<typename T,std::float_round_style R,typename U> T half_cast(U arg) { return
detail::half_caster<T,U,R>::cast(arg); }

```

```

    inline int feclearexcept(int excepts) { detail::errflags() &= ~excepts; return 0; }

    inline int fetestexcept(int excepts) { return detail::errflags() & excepts; }

    inline int feraiseexcept(int excepts) { detail::errflags() |= excepts; detail::raise(excepts); return 0; }

    inline int fegetexceptflag(int *flagp, int excepts) { *flagp = detail::errflags() & excepts; return 0; }

    inline int fesetexceptflag(const int *flagp, int excepts) { detail::errflags() = (detail::errflags()|(*flagp&excepts)) & (*flagp|~excepts); return 0; }

    inline void fethrowexcept(int excepts, const char *msg = "")
    {
        excepts &= detail::errflags();
        if(excepts & (FE_INVALID|FE_DIVBYZERO))
            throw std::domain_error(msg);
        if(excepts & FE_OVERFLOW)
            throw std::overflow_error(msg);
        if(excepts & FE_UNDERFLOW)
            throw std::underflow_error(msg);
        if(excepts & FE_INEXACT)
            throw std::range_error(msg);
    }
}


#undef HALF_UNUSED_NOERR
#undef HALF_CONSTEXPR
#undef HALF_CONSTEXPR_CONST
#undef HALF_CONSTEXPR_NOERR
#undef HALF_NOEXCEPT
#undef HALF_NOTHROW
#undef HALF_THREAD_LOCAL
#undef HALF_TWOS_COMPLEMENT_INT
#ifdef HALF_POP_WARNINGS
    #pragma warning(pop)
    #undef HALF_POP_WARNINGS
#endif

#endif
```

7.6 Light.hpp

```
//
//  NewLight.hpp
//  GL_Scene
//
//  Created by Alonso García on 13/1/25.
//

#pragma once

#include "glm.hpp"
#include <iostream>

#include "Cube.hpp"

namespace udit
{
    class Light {
    private:
        glm::vec3 position;

        glm::vec3 color;

        float ambientIntensity;

        float diffuseIntensity;

    public:
        Light(const glm::vec3& pos, const glm::vec3& col, float ambient, float diffuse);

        static std::shared_ptr <Light> make_light(const glm::vec3& pos, const glm::vec3& col, float ambient, float diffuse);

        void send_to_shader(GLuint program_id) const;
    };

}

```

7.7 Mesh.hpp

```
//
//  Mesh.hpp
//  GL_Geometry
//
//  Created by Alonso García on 11/12/24.
//

#pragma once

#include <vector>

#include "glm.hpp"
#include <gtc/matrix_transform.hpp>
#include <gtc/type_ptr.hpp>
#include "glad.h"

#include "Shader.hpp"

namespace udit
{
    enum class MeshType
    {
        BASIC,
        MESH,
        TERRAIN,
        SKYBOX
    };

    class Mesh
    {
    private:
        enum
        {
            COORDINATES_VBO,
            COLORS_VBO,
            NORMALS_VBO,
            INDEXES_VBO,
            TEXTURE_UV_VBO,
            VBO_COUNT
        };

        MeshType m_mesh_type;

    protected:
        std::vector<glm::vec3> coordinates;
        std::vector<glm::vec3> colors;
        std::vector<glm::vec3> normals;
        std::vector<GLuint> indices;
        std::vector<glm::vec2> texture_uvs;

        GLsizei number_of_vertices;

        void create_mesh(std::string mesh_name = "");

    private:
        GLuint vbo_ids[VBO_COUNT];
        GLuint vao_id;

        glm::mat4 model_view_matrix;
        glm::mat4 normal_matrix;

        std::shared_ptr < udit::Shader > m_shader;

    public:
        Mesh();

        Mesh(std::string & path);

        static std::shared_ptr <Mesh> make_mesh(MeshType type, const std::string &path = "");

        virtual ~Mesh();

        virtual void translate(glm::vec3 translation);

        virtual void rotate(glm::vec3 rotation, float angle);

        virtual void scale(glm::vec3 scale);

        virtual void update();

        virtual void render(glm::mat4 view_matrix);

        virtual void resize(glm::mat4 projection_matrix);

        virtual void set_shader(std::shared_ptr < udit::Shader > shader);

        GLuint get_shader_program_id() const;

        std::vector < GLint > get_shader_matrix_ids();

        glm::mat4 get_model_view_matrix() const { return model_view_matrix; }

        void set_model_view_matrix(glm::mat4 matrix) { model_view_matrix = matrix; }

        void set_mesh_type(MeshType type) { m_mesh_type = type; }
    };

}
```

7.8 Plane.hpp

```
//
//  Plane.hpp
//  GL_Geometry
//
//  Created by Alonso García on 11/12/24.
//

#pragma once

#include "glad.h"

#include "Mesh.hpp"

namespace udit
{
    class Plane : public Mesh
    {
    private:
        float width;

        float height;

        unsigned columns;

        unsigned rows;

    public:
        Plane();

        Plane(float size);

        Plane(float width, float height, unsigned columns, unsigned rows);

    private:
        void create_plane();
    };

}
```

7.9 GL_Scene/Scene.hpp File Reference

Clase que representa una escena 3D, gestionando objetos como el fondo, terreno, luz, etc.

```
#include <string>
#include "Shader.hpp"
#include "Light.hpp"
#include "Skybox.hpp"
#include "Plane.hpp"
```

Classes

- class udit::Scene

  Representa una escena 3D con un skybox, terreno, luz y otros elementos.

7.9.1 Detailed Description

Clase que representa una escena 3D, gestionando objetos como el fondo, terreno, luz, etc.

Esta clase es responsable de mantener y gestionar la escena 3D, incluyendo el fondo (skybox), el terreno, las luces y el resto de elementos gráficos. Permite actualizar, renderizar y redimensionar la escena, además de configurar las matrices de vista y proyección, y la luz del entorno.

7.10 Scene.hpp

Go to the documentation of this file.

```
//
//  Scene.hpp
//  GL_Geometry
//
//  Created by Alonso García on 9/12/24.
//

#pragma once

#include <string>
#include "Shader.hpp"
#include "Light.hpp"
#include "Skybox.hpp"
#include "Plane.hpp"

namespace udit
{
    class Scene
    {
    private:
        std::vector<std::string> skybox_faces =
        {
               "skybox_east.jpg", "skybox_west.jpg", "skybox_up.jpg",
               "skybox_down.jpg", "skybox_north.jpg", "skybox_south.jpg"
        };

        float angle = 0.0f;

        std::shared_ptr<Skybox> skybox;

        std::shared_ptr<Plane> terrain;

        std::shared_ptr<Plane> floor;

        std::shared_ptr<Mesh> bull;

        std::shared_ptr<Light> light;

        unsigned width, height;

        glm::mat4 view_matrix;

        glm::mat4 projection_matrix;

    public:
        Scene(unsigned width, unsigned height);

        void update();

        void render();

        void resize(unsigned width, unsigned height);

        void set_view_matrix(const glm::mat4& view);

        void set_projection_matrix(const glm::mat4& projection);

        void set_lights(GLuint shader_program_id);
    };
}
```

7.11 GL_Scene/Shader.hpp File Reference

Clase que representa un shader en OpenGL, gestionando la compilación y uso de programas de sombreado.

```
#include <iostream>
#include "glad.h"
#include "Texture.hpp"
```

Classes

- class udit::Shader

  Representa un shader program en OpenGL.

Enumerations

- enum class udit::ShaderType {
  SKYBOX , GEOMETRY , SINGLE_TEXTURE , TERRAIN ,
  DEFAULT }

  Enumeración que define los diferentes tipos de shaders.

7.11.1 Detailed Description

Clase que representa un shader en OpenGL, gestionando la compilación y uso de programas de sombreado.

La clase `Shader` gestiona la creación, compilación y uso de shaders en OpenGL, incluyendo tanto el vertex shader como el fragment shader. Además, permite la gestión de texturas asociadas al shader y la configuración de matrices para la proyección, vista y normales en el contexto de la cámara.

7.11.2 Enumeration Type Documentation

7.11.2.1 ShaderType

```
enum class udit::ShaderType  [strong]
```

Enumeración que define los diferentes tipos de shaders.

Define los tipos de shaders que la clase `Shader` puede usar para diferentes efectos visuales, como el skybox, geometría, textura única, terreno y por defecto.

Enumerator

SKYBOX	Shader para el skybox.
GEOMETRY	Shader para la geometría.
SINGLE_TEXTURE	Shader para una textura única.
TERRAIN	Shader para el terreno.
DEFAULT	Shader por defecto.

7.12 Shader.hpp

Go to the documentation of this file.

```
//
//  Shader.hpp
//  GL_Geometry
//
//  Created by Alonso García on 11/12/24.
//

#pragma once

#include <iostream>
#include "glad.h"
#include "Texture.hpp"

namespace udit
{
    enum class ShaderType
    {
        SKYBOX,
        GEOMETRY,
        SINGLE_TEXTURE,
        TERRAIN,
        DEFAULT
    };

    class Shader
    {
    private:
        GLuint program_id;

        ShaderType m_type;

        std::string m_name;

        std::string absolute_path =
"/Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/GL_Scene/";

        std::string m_vertex_default_source = absolute_path + "Shader_Default_Vertex.glsl";

        std::string m_fragment_default_source = absolute_path + "Shader_Default_Fragment.glsl";

        std::string m_vertex_source;

        std::string m_fragment_source;

        GLint  model_view_matrix_id;
        GLint  projection_matrix_id;
        GLint  normal_matrix_id;

        static const std::string    default_vertex_shader_code;
        static const std::string    default_fragment_shader_code;

        std::vector <std::shared_ptr<Texture» textures;

    public:
        Shader();

        Shader(ShaderType type, const std::string & vertex_source, const std::string &
fragment_source, const std::string & name);

        ~Shader();

        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 = ""
        );

        GLuint compile_shaders(const char * vertex_shader_code, const char * fragment_shader_code);

        GLint get_model_view_matrix_id() { return model_view_matrix_id; }

        GLint get_projection_matrix_id() { return projection_matrix_id; }

        GLint get_normal_matrix_id() { return normal_matrix_id; }

        GLuint get_program_id() const { return program_id; }

        void set_texture(const std::shared_ptr<Texture> & texture);

        void use() const;

        void set_texture_scale(float scale);

        bool has_textures() { return !textures.empty(); }

        void set_name(const std::string & name) { m_name = name; }

        std::string get_name() { return m_name; }

    private:
        void show_compilation_error(GLuint shader_id);

        void show_linkage_error(GLuint program_id);
    };
}
```

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.

```
//
//  Skybox.hpp
//  GL_Scene
//
//  Created by Alonso García on 21/12/24.
//

#pragma once

#include "Cube.hpp"
#include <vector>
#include <string>

namespace udit
{
    class Skybox : public Cube
    {
    private:
        unsigned int cubemapTexture;

        std::string filepath =
  "/Users/alonsoggdev/UDIT/Asignaturas/Programacion_Grafica/GL_Scene/resources/skybox/";

    public:
        Skybox();

        Skybox(float size, const std::vector<std::string>& faces);

        unsigned int getCubemapTexture() const { return cubemapTexture; }

    private:
        void loadCubemap(const std::vector<std::string>& faces);
    };
}
```

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.

```
//
//  Texture.hpp
//  GL_Scene
//
//  Created by Alonso García on 24/12/24.
//

#pragma once

#include <string>
#include <glad.h>

namespace udit
{
    enum class Texture_Type
    {
        COLOR,
        HEIGHT
    };

    class Texture
    {
    private:
        bool loaded = false;

        Texture_Type m_type;

    public:
        Texture(const std::string & path, GLenum texture_unit, Texture_Type type = Texture_Type::COLOR);

        ~Texture();

        void bind() const;

        void unbind() const;

        GLuint texture_id;

        GLenum texture_unit;

        std::string file_path;

        void load_texture();

        void set_type(Texture_Type type) { m_type = type; }

        bool is_loaded() { return loaded; }
    };
}

```

7.17 Window.hpp

```
//
//  Window.hpp
//  GL_Geometry
//
//  Created by Alonso García on 9/12/24.
//

#pragma once

#include <SDL.h>
```

```
#include <string>
#include <utility>

namespace udit
{

    class Window
    {
    public:

        enum Position
        {
            UNDEFINED = SDL_WINDOWPOS_UNDEFINED,
            CENTERED  = SDL_WINDOWPOS_CENTERED,
        };

        struct OpenGL_Context_Settings
        {
            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;
        };

    private:

        SDL_Window  * window_handle;
        SDL_GLContext opengl_context;

    public:

        Window
        (
            const std::string & title,
            int      left_x,
            int      top_y,
            unsigned width,
            unsigned height,
            const OpenGL_Context_Settings & context_details
        )
        :
            Window(title.c_str (), left_x, top_y, width, height, context_details)
        {
        }

        Window
        (
            const char * title,
            int      left_x,
            int      top_y,
            unsigned width,
            unsigned height,
            const OpenGL_Context_Settings & context_details
        );

       ~Window();

    public:

        Window(const Window & ) = delete;

        Window & operator = (const Window & ) = delete;

        Window(Window && other) noexcept
        {
            this->window_handle  = std::exchange (other.window_handle,  nullptr);
            this->opengl_context = std::exchange (other.opengl_context, nullptr);
        }

        Window & operator = (Window && other) noexcept
        {
            this->window_handle  = std::exchange (other.window_handle,  nullptr);
            this->opengl_context = std::exchange (other.opengl_context, nullptr);

            return  * this;
        }

    public:

        void swap_buffers ();

    };

}
```

Index