Haskell Implementation - Background (1A)

Young Won Lim
9/28/18

Please send corrections (or suggestions) to youngwlim@hotmail.com.
This document was produced by using LibreOffice.

Based on

The original work :

expression-oriented/cordic github
https://github.com/expression-oriented/cordic

This work is based on the work of Ben Barnes.

cordic source (1)

```
module CORDIC
(
  cordic
) where

import Util
```

https://github.com/expression-oriented/cordic

cordic source (2)

```
-- | (index, remainder, (x, y)) used in fold
type State = (Int,  Double, (Double, Double))

-- | Initialize (x, y) and index, execute fold, scale result
-- | Parameter `a` is the angle in radians, `n` is the number of iterations
-- | The result is a pair ( cos a, sin a )
cordic :: Double -> Int -> (Double, Double)
cordic a n = let
    initial = ( 0, a, (1, 0) )
    (i, _, (c, s)) = foldl step initial $ take n alist
    k = klist !! i
in ( k * c, k * s )
```

https://github.com/expression-oriented/cordic

cordic source (3)

```
-- |Core of the algorithm - generates next (x, y) from current
step :: State -> Double -> State
step (i, a, v) d
  | a > 0 = ( i', a - d, mult i   1  v )
  | a < 0 = ( i', a + d, mult i (-1) v )
  | otherwise = ( i', a, v )
where i' = i + 1
```

https://github.com/expression-oriented/cordic

cordic source (4)

```
-- | Multiplies 'vector' (x, y) by i'th rotation matrix
mult :: Int -> Double -> (Double, Double) -> (Double, Double)
mult i sign (x, y) = let
    mu = if sign < 0
      then negate
      else id
    x' = x - mu ( s y ( -i ))
    y' = y + mu ( s x ( -i ))
  in (x', y')
```

https://github.com/expression-oriented/cordic

cordic util (1)

```
{- Provides constants used by the CORDIC algorithm.
 - `alistF` is a list of angles [ atan 1, atan (1/2), atan (1/4, ... ]
 - `klistF` is a list of the scaling constants for each iteration
 - Traditionally these would have been hard-coded for performance; they are
 - generated programmatically here for simplicity.
 -}

module Util
(
  alist,
  klist
) where
```

https://github.com/expression-oriented/cordic

cordic util (2)

```
-- |Infinite list of angles with tangent ratios [1,  1/(2^i)]
alist :: [Double]
alist = [ atan ( 1 / 2 ^ e ) | e <- [ 0 .. ] ]
```

https://github.com/expression-oriented/cordic

cordic util (3)

```haskell
-- |Infinite list of scaling factors
klist :: [Double]
klist = klist' 1 ( k 0 )

-- |Recursive generator for scaling factors
klist' :: Int -> Double -> [Double]
klist' i n = n : klist' ( i + 1 ) (  k i * n )

-- |Scaling factor k at iteration i
k :: Int -> Double
k i = 1 / sqrt ( 1 + 2 ^^ (( -2 ) * i ))
```

https://github.com/expression-oriented/cordic

take

take :: Int -> [a] -> [a]

base Prelude, base Data.List

take **n**, applied to a list **xs**, returns the prefix of **xs** of length **n**,

or **xs** itself if n > length **xs**:

```
> take 5 "Hello World!" == "Hello"
> take 3 [1,2,3,4,5] == [1,2,3]
> take 3 [1,2] == [1,2]
> take 3 [] == []
> take (-1) [1,2] == []
> take 0 [1,2] == []
```

It is an instance of the more general Data.List.genericTake,

in which n may be of any integral type.

https://www.haskell.org/hoogle/?hoogle=take

shift

shift :: a -> Int -> a infixl 8

shift x i shifts x left by i bits if i is positive, or right by -i bits otherwise.
Right shifts perform sign extension on signed number types;
i.e. they fill the top bits with 1 if the x is negative and with 0 otherwise.

An instance can define either this unified shift or shiftL and shiftR,
depending on which is more convenient for the type in question.

https://www.haskell.org/hoogle/?hoogle=take

Numeric.Fixed

newtype Fixed

A signed 2s complement 15.16 scale fixed precision number

Constructors

Fixed

getFixed :: Cint

fromFixed :: Fixed -> Double

Source

toFixed :: Double -> Fixed

https://www.haskell.org/hoogle/?hoogle=take

!!

(!!) :: [a] -> Int -> a infixl 9

List index (subscript) operator, starting from 0.

It is an instance of the more general genericIndex,

which takes an index of any integral type.

!! indexes lists.

It takes a list and an index, and returns the item at that index.

If the index is out of bounds, it returns ⊥.

http://hackage.haskell.org/package/base-4.7.0.0/docs/Prelude.html#v%3a-33--33-

https://stackoverflow.com/questions/24421934/double-exclamation-marks-in-haskell

References

[1] https://github.com/expression-oriented/cordic