# include <stdlib.h>
# include <stdio.h>
# include <math.h>
# include <time.h>
# include "cordic.h"

/******************************************************************************/
void cordic ( double *x, double *y, double *z, int n )
/******************************************************************************/

CORDIC returns the sine and cosine using the CORDIC method.

Licensing:
This code is distributed under the GNU LGPL license.

Modified:
2012.04.17

Author:
Based on MATLAB code in a Wikipedia article.
Modifications by John Burkardt
Further modified by Young W. Lim

Parameters:

Input:
*x: x coord of an init vector
*y: y coord of an init vector
*z: angle (-90 <= angle <= +90)
n: number of iteration
    A value of 10 is low. Good accuracy is achieved
    with 20 or more iterations.

Output:
*xo: x coord of a final vector
*yo: y coord of a final vector
*zo: angle residue

Local Parameters:

Local, real ANGLES(60) = arctan ( (1/2)^(0:59) );
Local, real KPROD(33), KPROD(j) = product ( 0 <= i <= j ) K(i),
    K(i) = 1 / sqrt ( 1 + (1/2)^(2i) ).
*/
{
    # define ANGLES_LENGTH 60
    # define KPROD_LENGTH 33

double angle;
double angles[ANGLES_LENGTH] = {
    7.8539816339744830962E-01,
    4.6364760900080611621E-01,
    2.4497866312686415417E-01,
    1.2435499454676143503E-01,
    6.241880999595734874E-02,
    3.1239833430268276254E-02,
    1.562372862047683803E-02,
    7.812341060101112965E-03,
    3.9065301319669718276E-03,
    1.9531225164788186851E-03,
double c2;
double factor;
int j;
double kprod[KPROD_LENGTH] = {
  0.70710678118654752440,
  0.63245553203637586640,
  0.61357199107789634961,
  0.60883391251775242102,
  0.60764825625616820093,
  0.60735177014129593034,
  0.60727764409352599995,
  0.60725911216029741373,
  0.60725447932161602974,
  0.60725332105948137373,
  0.60725303152913433540,
  0.60725295917292823792,
  0.60725294104139716351,
  0.60725293651701023413,
  0.6072529351031931731,
double pi = 3.141592653589793;

double poweroftwo;

double s2;

double sigma;

double sign_factor;

double theta;

double xn, yn;

/*
 Initialize loop variables:
*/
theta = *z;

xn = *x;
yn = *y;

poweroftwo = 1.0;
angle = angles[0];

/*
 Iterations
*/
for ( j = 1; j <= n; j++ )
{
    if ( theta < 0.0 )
    {
        sigma = -1.0;
    }
    else
    {
        sigma = 1.0;
    }

    factor = sigma * poweroftwo;

    *x = xn - factor * yn;
    *y = factor * xn + yn;

    xn = *x;
    yn = *y;

    /*
 Update the remaining angle.
*/
    theta = theta - sigma * angle;

    poweroftwo = poweroftwo / 2.0;

    /*
 Update the angle from table, or eventually by just dividing by two.
*/
    if ( ANGLES_LENGTH < j + 1 )
{ angle = angle / 2.0;
}
else
{ angle = angles[j];
}
*z = theta;

/*
Adjust length of output vector to be \([\cos(\beta), \sin(\beta)]\)

KPROD is essentially constant after a certain point, so if \(N\) is large, just take the last available value.
*/
if ( 0 < n )
{
    *c = *c * kprod [ i4_min ( n, KPROD_LENGTH ) - 1 ];
    *s = *s * kprod [ i4_min ( n, KPROD_LENGTH ) - 1 ];
}

/*
Adjust for possible sign change because angle was originally not in quadrant 1 or 4.
*/
* c = sign_factor * *c;
* s = sign_factor * *s;

return;

# undef ANGLES_LENGTH
# undef KPROD_LENGTH
}