```
::::::::::::::
cordic_tb01.cpp
::::::::::::::
# include <cstdlib>
# include <cmath>
# include <iostream>
# include <iomanip>
# include <fstream>

using namespace std;

# include "Core.hpp"
# include "Angles.hpp"

string GnuTerm;



//****************************************************************************
//   Purpose:
//
//     Explore Angles Space using Class Angles
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.02.05
//
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//****************************************************************************



int main (int argc, char * argv[])
{

  double pi = 3.141592653589793;
  double x, y, z;
  int    nBreak     = 0;    // number of such breaking events
  int    nBreakInit = 0;    // initialize the nBreak counter
  char   path[256]  = "";   // path string in the binary angle tree

  // ----------------------------------------------------------------
  // nIter      : Number of Iteration = Height of binary angle tree
  // nAngle     : Number of Angles    = Number of Leaf Nodes
  // th         : threshold for breaking the cordic algorithm's loop
  // ----------------------------------------------------------------
  int    nIter      = 20;
  int    nAngle     = 1 << nIter;
  double th         = 0.0;
  // ----------------------------------------------------------------
  // GnuTerm    : for gnuplot (wxt: monitor, emf: file)
  // nSamples   : determines the number of uniform samples over [-pi/2, +pi/2]
  // plotEn     : enable plotting
  // ----------------------------------------------------------------
```

```
        GnuTerm   = "wxt";
  int   nSamples  = 1000;
  int   plotEn    = 1;
  // ------------------------------------------------------------------
  // useTh      : thresholding
  // useThDisp  : display thresholding statistics
  // useATAN    : use atan() instead angles array values
  // ------------------------------------------------------------------
  int   useTh     =0;
  int   useThDisp =0;
  int   useATAN   =0;


//======================================================================
# include "cordic_tb01.param.cpp"
//======================================================================

  cout << "cordic_tb01 parameters " << endl;
  cout << "----------------------------------------------------------\n";
  cout << "   nIter      = " << nIter      << endl;
  cout << "   nAngle     = " << nAngle     << endl;
  cout << "   th         = " << th         << endl;
  cout << "----------------------------------------------------------\n";
  cout << "   GnuTerm    = " << GnuTerm    << endl;
  cout << "   nSamples   = " << nSamples   << endl;
  cout << "   plotEn     = " << plotEn     << endl;
  cout << "----------------------------------------------------------\n";
  cout << "   useTh      = " << GnuTerm    << endl;
  cout << "   useThDisp  = " << nSamples   << endl;
  cout << "   useATAN    = " << plotEn     << endl;
  cout << "----------------------------------------------------------\n";



//======================================================================
// # include "cordic_check.cpp"
//======================================================================


  //----------------------------------------------------------------------
  // x = 1.0, y = 0.0, z = [-pi/2, +pi/2], step = pi/(2*nSamples)
  //----------------------------------------------------------------------
  FILE * fp;
  int i;

  double cosz, sinz;
  double max_err=0.0, max_errn=0.0;
  double xx=0.0, yy=0.0, zz=0.0;
  double sum_xx =0.0, sum_xx2 =0.0;
  double sum_yy =0.0, sum_yy2 =0.0;
  double sum_xx_n =0.0, sum_xx2_n =0.0;
  double sum_yy_n =0.0, sum_yy2_n =0.0;
  int    cnt_xx =0, cnt_yy =0;


  Angles AllAngles(nIter, 2*nAngle-1);

  /* 3  */ AllAngles.calc_statistics();

  th = AllAngles.avg_delta;
  // th = (AllAngles.max_angle - AllAngles.max_angle) / AllAngles.getNAngle();

  Core C;

  C.setUseTh(useTh);
  C.setUseThDisp(useThDisp);
  C.setUseATAN(useATAN);
```

```
C.setLevel(nIter);
C.setThreshold(th);

cout << "cordic core parameters " << endl;
cout << "-------------------------------------------------------\n";
cout << "   useTh     = " << C.getUseTh()      << endl;
cout << "   useThDisp = " << C.getUseThDisp()  << endl;
cout << "   useATAN         = " << C.getUseATAN()           << endl;
cout << "-------------------------------------------------------\n";
cout << "   level           = " << C.getLevel()             << endl;
cout << "   threshold       = " << C.getThreshold()         << endl;
cout << "-------------------------------------------------------\n";


//-----------------------------------------------------------------------
// I=0: finding max_err & max_errn
// I=1: writing scaled data into files
//-----------------------------------------------------------------------
for (int I=0; I<2; ++I) {
//-----------------------------------------------------------------------
C.setNBreak(nBreak=0);
C.setNBreakInit(nBreakInit=0);

if (I==1) fp = fopen("test.dat", "w+");

for (i=-nSamples; i<=nSamples; ++i) {
  x = 1.0;
  y = 0.0;
  z = zz = (pi / (2*nSamples)) * (i);

  cosz = cos(z);
  sinz = sin(z);

  C.setNBreakInit(nBreakInit++);
  //.........................................................
  C.cordic(&x, &y, &z);
  //.........................................................

  xx = (x-cosz);
  yy = (y-sinz);

  if (I==0) {
    sum_xx += xx; sum_xx2 += (xx*xx);
    sum_yy += yy; sum_yy2 += (yy*yy);

    if (max_err < fabs(xx)) max_err = fabs(xx);
    if (max_err < fabs(yy)) max_err = fabs(yy);
    if (fabs(cosz) > 1.0e-10) {
       if (max_errn < fabs(xx/cosz))
           max_errn = fabs(xx/cosz);
       sum_xx_n += xx/cosz;
       sum_xx2_n += (xx*xx)/(cosz*cosz);
       cnt_xx++;
    }
    if (fabs(sinz) > 1.0e-10) {
       if (max_errn < fabs(yy/sinz))
           max_errn = fabs(yy/sinz);
       sum_yy_n += yy/sinz;
       sum_yy2_n += (yy*yy)/(sinz*sinz);
       cnt_yy++;
    }
  } else {
    fprintf(fp, "%f", zz);                          // col(1)
    fprintf(fp, " %f %f ", cosz, sinz);             // col(2,3)
    fprintf(fp, " %f %f ", x, y);                   // col(4,5)
    fprintf(fp, " %g %g ", xx/max_err, yy/max_err);   // col(6,7)
```

```
    xx /= cosz;
    yy /= sinz;
    fprintf(fp, " %g %g ", xx/max_errn, yy/max_errn); // col(8,9)
    fprintf(fp, " \n");
  }

} /* end of i */


if (I==0) {
  cout << "max_err  = " << max_err  << endl;
  cout << "max_errn = " << max_errn << endl;
  double avg = 0.0, mse = 0.0, rms =0.0;
  cout << "....................................................\n";
  avg = sum_xx / (2*nSamples+1);
  mse = sum_xx2 / (2*nSamples+1);
  rms = sqrt(mse);
  rms = sum_xx2;
  cout << "E[(x-cosz)]               : cos err avg = " <<  avg << endl;
  cout << "E[(x-cosz)^2]             : cos err mse = " <<  mse << endl;
  cout << "sqrt{E[(x-cosz)^2]}       : cos err rms = " <<  rms << endl;
  cout << "....................................................\n";
  avg = sum_yy / (2*nSamples+1);
  mse = sum_yy2 / (2*nSamples+1);
  rms = sqrt(mse);
  cout << "E[(y-sinz)]               : sin err avg = " <<  avg << endl;
  cout << "E[(y-sinz)^2]             : sin err mse = " <<  mse << endl;
  cout << "sqrt{E[(y-sinz)^2]}       : sin err rms = " <<  rms << endl;
  cout << "....................................................\n";
  avg = sum_xx_n / cnt_xx;
  mse = sum_xx2_n / (cnt_xx*cnt_xx);
  rms = sqrt(mse);
  cout << "E[(x-cosz)/cosz]          : cos nerr avg = " << avg << endl;
  cout << "E[(x-cosz)/cosz}^2]       : cos nerr mse = " << mse << endl;
  cout << "sqrt{E[(x-cosz)/cosz}^2]} : cos nerr rms = " << rms << endl;
  cout << "....................................................\n";
  avg = sum_yy_n / cnt_yy;
  mse = sum_yy2_n / (cnt_yy*cnt_yy);
  rms = sqrt(mse);
  cout << "E[(y-sinz)/sinz]          : sin nerr avg = " << avg << endl;
  cout << "E[(y-sinz)/sinz}^2]       : sin nerr mse = " << mse << endl;
  cout << "sqrt{E[(y-sinz)/sinz}^2]} : sin nerr rms = " << rms << endl;
} else {
  fclose(fp);
}


//-------------------------------------------------------------------------
}  /* end of I */
//-------------------------------------------------------------------------



if (plotEn ==0) return 0;

//-------------------------------------------------------------------------
// ** GnuTerm ** MUST Be set
//-------------------------------------------------------------------------
ofstream myout;

int nemf = (GnuTerm.compare("emf") != 0);

// writing gnuplot commands
myout.open("command.gp");
myout << "set terminal " << GnuTerm << endl;
```

```
myout << "set xlabel \"uniform scaled angles\" " << endl;
myout << "set ylabel \"error using (x, cosz) or (y, sinz)\" " << endl;
myout << "set yrange [-1.2:+1.2]" << endl;

myout << "set output 'tb01.error.cos.emf'" << endl;
myout << "set title \"cos error plot ";
myout << "(max_err=" << max_err << ")\"" << endl;

myout << "plot 'test.dat' using 1:2 w lines,  ";
myout << "     'test.dat' using 1:4 w lines,  ";
myout << "     'test.dat' using 1:6 w lines  ";
myout << endl;
if (nemf) myout << "pause mouse keypress" << endl;

myout << "set output 'tb01.error.sin.emf'" << endl;
myout << "set title \"sin error plot ";
myout << "(max_err=" << max_err << ")\"" << endl;

myout << "plot 'test.dat' using 1:3 w lines,  ";
myout << "     'test.dat' using 1:5 w lines,  ";
myout << "     'test.dat' using 1:7 w lines  ";
myout << endl;
if (nemf) myout << "pause mouse keypress" << endl;

myout << "set output 'tb01.error.all.emf'" << endl;
myout << "set title \"cos, sin error plot ";
myout << "(max_err=" << max_err << ")\"" << endl;

myout << "plot 'test.dat' using 1:2 w lines,  ";
myout << "     'test.dat' using 1:3 w lines,  ";
myout << "     'test.dat' using 1:4 w lines,  ";
myout << "     'test.dat' using 1:5 w lines,  ";
myout << "     'test.dat' using 1:6 w lines,  ";
myout << "     'test.dat' using 1:7 w lines   ";
myout << endl;
if (nemf) myout << "pause mouse keypress" << endl;


myout << "set output 'tb01.errorn.cos.emf'" << endl;
myout << "set title \"cos normalized error plot ";
myout << "(max_err=" << max_errn << ")\"" << endl;

myout << "plot 'test.dat' using 1:2 w lines,  ";
myout << "     'test.dat' using 1:4 w lines,  ";
myout << "     'test.dat' using 1:8 w lines  ";
myout << endl;
if (nemf) myout << "pause mouse keypress" << endl;

myout << "set output 'tb01.errorn.sin.emf'" << endl;
myout << "set title \"sin normalized error plot ";
myout << "(max_err=" << max_errn << ")\"" << endl;

myout << "plot 'test.dat' using 1:3 w lines,  ";
myout << "     'test.dat' using 1:5 w lines,  ";
myout << "     'test.dat' using 1:9 w lines  ";
myout << endl;
if (nemf) myout << "pause mouse keypress" << endl;

myout << "set output 'tb01.errorn.all.emf'" << endl;
myout << "set title \"cos, sin normalized error plot ";
myout << "(max_err=" << max_errn << ")\"" << endl;

myout << "plot 'test.dat' using 1:2 w lines,  ";
myout << "     'test.dat' using 1:3 w lines,  ";
myout << "     'test.dat' using 1:4 w lines,  ";
myout << "     'test.dat' using 1:5 w lines,  ";
```

```
  myout << "     'test.dat' using 1:8 w lines,  ";
  myout << "     'test.dat' using 1:9 w lines   ";
  myout << endl;
  if (nemf) myout << "pause mouse keypress" << endl;




  myout.close();


  system("gnuplot command.gp");



  return 0;

}
::::::::::::::
cordic_tb01.param.cpp
::::::::::::::
  // ---------------------------------------------------------------
  // nIter      : Number of Iteration = Height of binary angle tree
  // nAngle     : Number of Angles    = Number of Leaf Nodes
  // th         : threshold for breaking the cordic algorithm's loop
  // ---------------------------------------------------------------
  // GnuTerm    : for gnuplot (wxt: monitor, emf: file)
  // nSamples   : determines the number of uniform samples over [-pi/2, +pi/2]
  // plotEn     : enable plotting
  // ---------------------------------------------------------------
  // UseTh      : flags for thresholding
  // UseThDisp  : flags for displaying threshold statistics
  // useATAN    : flags for using atan() function
  // ---------------------------------------------------------------

  nIter        = 12;
  nAngle       = 1 << nIter;
  th           = 0.001;

  GnuTerm      = "wxt";  // wxt or emf
  nSamples     = 1000;
  plotEn       = 1;

  useTh        = 0;
  useThDisp    = 0;
  useATAN      = 0;


  cout << "-------------------------------------------------------\n";
  cout << "cordic_tb01  "                 << endl;
  cout << "  [nIter]     : atoi(argv[1]) " << endl;
  cout << "  [th]        : atof(argv[2]) " << endl;
  cout << "  [GnuTerm]   :     (argv[3]) " << endl;
  cout << "  [nSamples]  : atoi(argv[4]) " << endl;
  cout << "  [plotEn]    : atoi(argv[5]) " << endl;
  cout << "  [useTh]     : atoi(argv[6]) " << endl;
  cout << "  [useThDisp] : atoi(argv[7]) " << endl;
  cout << "  [useATAN]   : atoi(argv[8]) " << endl;
  cout << "-------------------------------------------------------\n";


  if (argc > 1 ) {
    nIter = atoi(argv[1]);
    nAngle = 1 << nIter;
  }

  if (argc > 2) {
```

```
    th = atof(argv[2]);
  }

  if (argc > 3) {
    GnuTerm = argv[3];
  }

  if (argc > 4) {
    nSamples = atoi(argv[4]);
  }

  if (argc > 5) {
    plotEn = atoi(argv[5]);
  }

  if (argc > 6) {
    useTh = atoi(argv[6]);
  }

  if (argc > 7) {
    useThDisp = atoi(argv[7]);
  }

  if (argc > 8) {
    useATAN = atoi(argv[8]);
  }


::::::::::::::
cordic_check.cpp
::::::::::::::
  Core TC;

  TC.setLevel(20);
  TC.setThreshold(0.0);
  TC.setNBreak(0);
  TC.setNBreakInit(0);


  cout << endl << endl;
  cout << "* Basic Testing \n";
  cout << "  [1, 0, 0   ] -> [1.0         0.0,        0] \n";
  cout << "  [1, 0, pi/6] -> [0.86602540, 0.50000000, 0] \n";
  cout << "  [1, 0, pi/4] -> [0.70710678, 0.70710678, 0] \n";
  cout << "  [1, 0, pi/3] -> [0.50000000, 0.86602540, 0] \n";
  cout << endl << endl;


  //------------------------------------------------------------------------
  // printf ("\nGrinding on [K, 0, 0]\n");
  // Circular (X0C, 0L, 0L);
  //------------------------------------------------------------------------
  x = 1.0;
  y = 0.0;
  z = 0.0;
  cout << "-----------------------------------------------\n"
       << "xi=" << x << " yi=" << y << " zi=" << z << "\n";

  //.........................................................
  TC.cordic(&x, &y, &z);
  //.........................................................

  cout  << "xo=" << x << " yo=" << y << " zo=" << z << "\n";


  //------------------------------------------------------------------------
  // printf ("\nGrinding on [K, 0, pi/6] -> [0.86602540, 0.50000000, 0]\n");
```

```
// Circular (X0C, 0L, HalfPi / 3L);
//-----------------------------------------------------------------------
x = 1.0;
y = 0.0;
z = pi / 6.0;
cout << "-----------------------------------------------\n"
     << "xi=" << x << " yi=" << y << " zi=" << z << "\n";

//.....................................................
TC.cordic(&x, &y, &z);
//.....................................................

cout  << "xo=" << x << " yo=" << y << " zo=" << z << "\n";



//-----------------------------------------------------------------------
// printf ("\nGrinding on [K, 0, pi/4] -> [0.70710678, 0.70710678, 0]\n");
// Circular (X0C, 0L, HalfPi / 2L);
//-----------------------------------------------------------------------
x = 1.0;
y = 0.0;
z = pi / 4.0;
cout << "-----------------------------------------------\n"
     << "xi=" << x << " yi=" << y << " zi=" << z << "\n";

//.....................................................
TC.cordic(&x, &y, &z);
//.....................................................

cout  << "xo=" << x << " yo=" << y << " zo=" << z << "\n";



//-----------------------------------------------------------------------
// printf ("\nGrinding on [K, 0, pi/3] -> [0.50000000, 0.86602540, 0]\n");
// Circular (X0C, 0L, 2L * (HalfPi / 3L));
//-----------------------------------------------------------------------
x = 1.0;
y = 0.0;
z = pi / 3.0;
cout << "-----------------------------------------------\n"
     << "xi=" << x << " yi=" << y << " zi=" << z << "\n";

//.....................................................
TC.cordic(&x, &y, &z);
//.....................................................

cout  << "xo=" << x << " yo=" << y << " zo=" << z << "\n";
```