Pakistan Journal of Information and Technology 1(2): 136-141, 2002

Optimizing Speed and Accuracy of Trigonometric Calculations with Real Numbers

Muhammad Zaheer Aziz and Khalid Rashid
Department of Computer Science, International Islamic University
Islamabad, Pakistan

Abstract: A wide range of scientific areas requires calculation of trigonometric ratios like sine and cosine repeatedly in their computations. The calculation of these trigonometric ratios is implemented in the library functions of programming languages using Maclaurin's series which is a processor hungry technique. This paper aims for introducing a method to calculate the trigonometric ratios in an efficient manner as compared to available technique. The proposed method of storing the results of sine and cosine functions for all integer and fractional angles into separate arrays. In order to calculate the sine or cosine value for a particular angle the integer part is considered as α and the fractional part as β. The trigonometric relations for Sin ($\alpha + \beta$) and Cos ($\alpha + \beta$) are used to combine the result of integer and fractional parts. The outcome of the new method was tested in terms of accuracy, storage space, and efficiency using a program written in C++. It was found that the proposed method provides 100% accuracy, saves storage space, and is faster as compared to the existing methods.

Key Words: Sine Calculation, Cosine Calculation, Speed Optimization, Space Optimization

Introduction

The calculation of trigonometric ratios like Sine and Cosine is a processor hungry task. There are many areas of science and computing in which use of such calculations is heavily involved. These areas include engineering applications, astronomy, space science, graphics, image processing, computer vision, machine control, and virtual reality etc. Equations of these fields not only contain complex use of the trigonometric functions but also require iterations of the computations for thousands of times (Stevens and Christopher, 1993; Yaguchi and Springer-VBerlag, 1991). On the platform of the even latest personal computers such processing takes a considerably long time. Even for larger machines optimizing the speed of computation will lead to quicker solutions to problems.
As an example we can consider rotation of a still image having 100 x 100. This image would occupy a fairly small area of the computer screen. Actual images that can be used to convey reasonable information are quite large. Our sample image has 10,000 pixels in it. The coordinates of the transformed pixel (x^t, y^t) after rotation of a single pixel from the original coordinates (x, y) will be obtained from the equations :

$$x^t = x_o + (x - x_o) \cos\theta - (y - y_o) \sin\theta$$
$$y^t = y_o + (x - x_o) \sin\theta - (y - y_o) \cos\theta$$

where θ is the angle of rotation and (x_o, y_o) is the center of rotation (Heiny, 1994). We can see that it involves calculation of trigonometric functions four times. Repeating the same process for the whole 100 x 100 image would require 40,000 calls to the functions. This was a very simple example to elaborate the problem. The number of calculations required are very large for other similar objects like animated images , 3D figures, and video input. Especially in case of computer vision lot of transformations are applied on the still image or video camera input. In these situations there are many objects moving or rotating in different directions and different angles hence significantly increasing processing time (Shah, 2000).

Materials and Methods

The simplest method to solve the equations involving trigonometric ratios is the use of built-in library functions for the programming languages like Fortran, Pascal, C, and C++. In case the function call is needed for one or few previously known angles then the results are saved in one or more variables and these are used in equations rather than calculating the value each time. On the other hand if a large number of integer angles are involved then one way to deal with it is to save the sines of all the integer angles from 0 to 360 in an array and access the array instead of calculating the series each time. Description of these methods is given below :

Library Function Calls: The library functions available in the programming languages use the Maclaurin's Series to find the value of sine and cosine for a given angle. These functions take a real number as parameter for the angle measured in radians (Turbo; MSDN). Following Maclaurin's Series is used for finding the values for the provided angle A (Anton, 1995).

$$\text{Sin A} = A - \frac{A^3}{3!} + \frac{A^5}{5!} - \frac{A^7}{7!}$$

$$\text{Cos A} = 1 - \frac{A^2}{2!} + \frac{A^4}{4!} - \frac{A^6}{6!}$$

The library functions find out all the exponents and factorials and sum up the terms up to a required accuracy. This requires a significant amount of computing time.

Stored Values Method: The call to library functions each time when needed is feasible only when there is a requirement of trigonometric ratios for a very small number of times. There are situations when a single or a small number of results of trigonometric functions is needed repeatedly for many equations or in a loop of the computer program. In this case the required values are stored in variables and they are used in equations instead of calling the functions repeatedly (Stevens, 1993). For example if we want to rotate an image about an angle of 30 degrees then we store sine and cosine of 30 in two variables. Now the same values will be used for all pixels of the image in the loop. Hence result available in the variables is directly used rather than calculating the series each time. This reduces computation time substantially.

Array Access Method: The method mentioned in section 2.2 speeds up the computations when there is a need of sine and cosine values for a very small number of angles. In case the angles cannot be predicted in advance then the said method fails to help. For example when a 3D object is viewed in Virtual Reality environment in a VRML browser (Charlie Morey) or a 3D editor like 3D Studio Max (Mathew Klare) then the angle of rotation or direction of view is taken as input from the user at runtime. Hence it varies continuously by pressing a key or mouse button. Moreover if the environment comprises of animated objects then each object uses a different angle for computation of its coordinates.

As far as integer angles are concerned the sines and cosines of all angles from 0 to 360 degrees can be stored in an array (Aziz and Khalid, 2000). Taking advantage of the nature of sine and cosine functions that Cos (90 + X) = Sin X we can conserve storage space and make only one array in which sines of angles from 0 to 360 degrees may be stored. To find the value of sine for an angle X we will access the array at index X and for finding the cosine we will access the element at index (90 + X) ⊕ 360 in the same array. Here ⊕ means remainder after division. For example we need cosine for angle 300° then the index to access from the array will be 30 which means we access value stored for Sine of 30 in the array. This method reduces computation time by 80% without loosing the accuracy the calculations are more that 4 times faster than calling the library functions repeatedly.

Proposed Method: The array access method mentioned in section 2.3 is the most suitable for angles having integer values form 0° to 360°. Many scientific computations need results of sine and cosines for angles in real numbers having fractional as well as integer part. For such applications the said methodology has to be extended so that trigonometric ratios may be available for angles other than just integers. Conversion of the system to cope with real angle involves certain issues which are discussed below:

Issues and Problems: The main issue in dealing with real angles is building of an array that stores results for all the fractional and integer angles. If we store all the integer and fractional values then the size of array will grow substantially large. For example, if we want to provide an array having results of Cosines for all real angles from 0.0° to 360.0° with fractional part from 0.1° to 0.9° then the array size will be 360 x 10 i.e. 3600 elements. If we need more accuracy in the angle say up to 1/100 of a degree then the array size will grow to 360 x 100 which is 36,000 elements.

The other important issue is to access the array in order to find the value of trigonometric ratio for a particular angle. Suppose we make a single array that holds all the possible values that may vary in size for increasing accuracy. For such an array we have to change the expression that will calculate the index that points to the required element of the array.

The Solution: Keeping in view the above mentioned issues a solution has to be designed which can tolerate the change in accuracy of fractional part of the angle and hence access method remains the same for different levels of accuracy. For this purpose we take advantage of the mathematical relationships :

$\sin(\alpha + \beta) = \cos\alpha . \sin\beta + \sin\alpha . \cos\beta$

$\cos(\alpha + \beta) = \cos\alpha . \cos\beta - \sin\alpha . \sin\beta$

Taking integer part of the angle as α and fractional part as β the solution to the problem becomes fairly simple. We can make two different arrays, one for the integer angles from 0° to 360° and the other for fractional part say from 0.001° to 0.999°. Now we have 360 + 1000 = 1360 elements of arrays that can deal with the accuracy of angles from 0.000° to 359.999°. If we make an array for holding the values for all the involved angles for the same accuracy, the array size would have been 360 * 1000 = 360000 which is obviously not feasible.

Using the same array for storing and accessing results of both sine and cosine ratios as mentioned in section 2.3 will be complex task in this case. Conversion of integer part as well as fractional part for the other ratio will be computationally heavy and will reduce the advantage of processing time. Hence two sets of arrays are used one for Sine and other for Cosine.

Fig.1 shows the algorithm that creates the said four arrays. The first loop creates arrays for integer angles for both Sine and Cosine functions while the second loop is used for creating the array for fractional part of the array.

Getting the Results: The results of the trigonometric ratios will be found using the mathematical relation mentioned in section 3.2. The index to access the array for integer degrees will be the same as the angle itself and the index to access the result for a fractional angle will be 1000 x Fractional Angle for angles having accuracy of 1/1000 degrees. Equations for finding the index for fractional part, value of Sine, and value of cosine for a real angle having integer part Ang and fractional part FracAng are given below :

```
//Storing integer angle values in array
for Ang = 0 to Ang<=360 do
Begin
        SIN[Ang]=sin(Ang*PiBy180);
        COS[Ang]=cos(Ang*PiBy180);
End

//Storing Real angle values from 0.001 to
0.999
i=0;
for FracAng=0 to 0.999 do)
Begin
        SinFrac[i]=sin(FracAng*PiBy180);
        CosFrac[i]=cos(FracAng*PiBy180);
        i=i+1;
        FracAng=FracAng+0.001;
End
```

Fig. 1: Algorithm to Create the Arrays

```
FracAngIndex=1000*FracAng;
SineValue=COS[Ang]*SinFrac[FracAngIndex]+
        SIN[Ang]*CosFrac[FracAngIndex];
CosineValue=COS[Ang]*CosFrac[FracAngIndex]-
        SIN[Ang]*SinFrac[FracAngIndex];
```

As an example if we need to find value of Sin (30.163) then the result will be found using the array access as follows :

```
FracAngIndex=1000*0.163;
⇒ FracAngIndex=163
SineValue=COS[30]*SinFrac[163]+
        SIN[30]*CosFrac[163];
```

Analysis of Results: The results of the proposed method have to be analyzed in terms of accuracy, usage of storage space, and speed of processing. For this analysis a complete program has been developed in C++ and results are produced using the proposed method. The related existing methods are also used for producing the same results in order to compare the performance of the new method. A detailed analysis for the above mentioned three criteria of judgement is given below :

Accuracy of Results: There are always chances of numerical errors dealing with real numbers in calculations like multiplication and division in a computer. These errors occur due to round off and exceeding of storage bounds. Use of high precision variables that allow storage of real numbers with greater accuracy can minimize such errors. All scientific programming languages provide data types that allow storage of high precision real numbers. In C++ the high precision data type is named *double* while for low precision real numbers it has the data type named float.

In order to preserve accuracy it is highly recommended that double type variable be used for the method proposed in this paper. For testing the accuracy of the proposed method the results obtained from array access and by calling the library function of sine and cosine for a large set of angles are stored in a file and graphs are plotted to identify any differences. The code used for this purpose is shown in Fig. 2.

```
for (Ang = 0; Ang<=360;Ang++)
{
  for
(FracAng=0;FracAng<=0.999;FracAng+=0.001)
  {
    rad=(Ang+FracAng)*PiBy180;
    fprintf(fp, "%f, ", Ang+FracAng);
    FracAngIndex=1000*FracAng;
    //Getting Sin(x) using library function
    Value=sin(rad);
    fprintf(fp, "%f, ", Value);
    //Getting Sin(x) using array access
    Value=COS[Ang]*SinFrac[FracAngIndex]+
        SIN[Ang]*CosFrac[FracAngIndex];
    fprintf(fp, "%f, ", Value);
    //Getting Cos(x) using library function
    Value=cos(rad);
    fprintf(fp, "%f, ", Value);
    //Getting Cos(x) using array access
    Value=COS[Ang]*CosFrac[FracAngIndex]-
        SIN[Ang]*SinFrac[FracAngIndex];
    fprintf(fp, "%f\n", Value);
  }
}
```

Fig. 2: Code for Testing Accuracy

The code given in Fig. 2 produces a very large set of data comprising of 360 x 1000 = 360000 sets of values. In order to show the outcome we may take subsets of these values and plot graphs to see if the results are accurate. Fig. 3 shows the graph of results obtained from calling the sine and cosine library function for angles from 0.000 ° to 360.000° with an interval of 3.333°. Fig. 4 shows the graph of the values obtained by using the proposed array access method with the same set of angles as input. The results show that there is no difference in values obtained from both methods. So there is no loss of accuracy.

Usage of Storage Space: We compare the usage of storage space in computer's memory for the two options of saving the results. First option is to build a single array that stores all the results for angles starting from 0 to 360 with a fractional increment. The other option is to make separate arrays for integer angles and fractional part of the angles. Table – 1 shows that the option of using two arrays adopted in this paper is far better in usage of storage space.

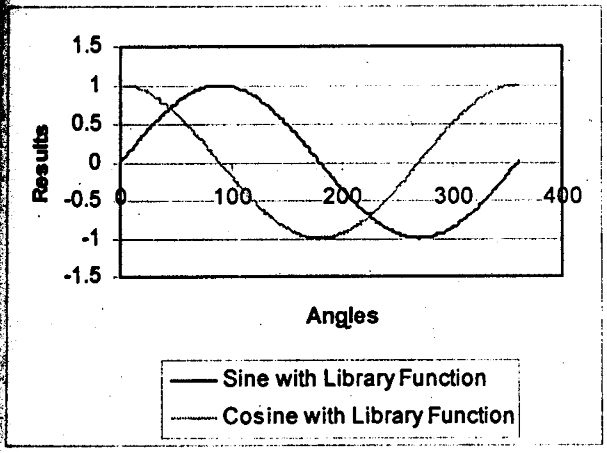

Fig. 3: Results of Sine and Cosine using Library Function Calls

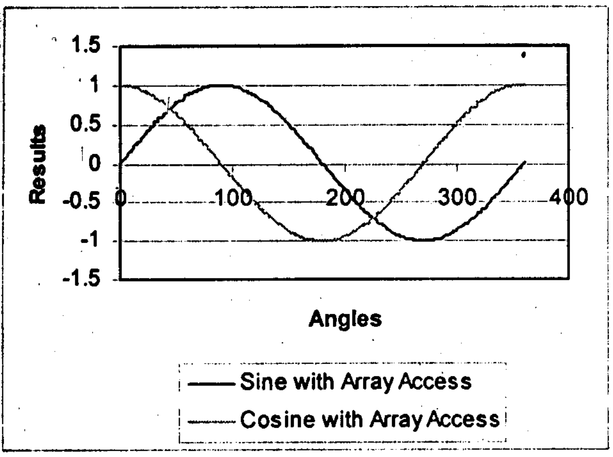

Fig. 4: Results of Sine and Cosine using Array Access

Table 1: Number of Array Elements Needed to Store Results

Angle Precision	Single Array	Two Arrays
0.1 - 0.9	3600	370
0.01 - 0.99	36000	460
0.001 - 0.999	360000	1360

Time for Processing: The main objective of this paper is to introduce a method to calculate the trigonometric ratios in an efficient manner as compared to available techniques. In order to find the time required for processing in both methods values of sine and cosine are computed for all the angles from 0.000° to 360.000° with an increment of 0.001°. This makes 360000 calculations in all. The time required for this processing is recorded for both methods of library function and array access. This process is repeated for different number of repetitions to know the time effect for increase in number of calculations. Fig. 5 and Fig. 6 show the algorithms used to obtain the time data for sine and cosines using available library access method and proposed array access method respectively. The data is then plotted in a graph to compare the outcome. Fig. 7 and Fig. 8 clearly show that the method introduced in this paper has an edge over the existing method.

```
//Processing time of sin function
//for N x 360 x 1000 calculations
for N = 1 to 29 step 2 do
Begin
  T1 = gettime();
  for i = 0 to N
  Begin
    Ang=0;
    while (Ang<360)
    Begin
          for FracAng=0 to 0.999 do
          Begin
            Rad=(Ang+FracAng) x PiBy180;
            Value=sin(rad);
            FracAng+=0.001;
          End
          Ang=Ang+1;
    End
  End
  T2 = gettime();
  Time_Interval=T2 - T1; //unit is 1/100 sec
  Write(Time_Interval);
End

//Processing time of cos function
//for N x 360 x 1000 calculations
for N = 1 to 29 step 2 do
Begin
  T1 = gettime();
  for i = 0 to N
  Begin
    Ang=0;
    while (Ang<360)
    Begin
          for FracAng=0 to 0.999 do
          Begin
                rad=(Ang+FracAng)x PiBy180;
                Value=cos(rad);
                FracAng+=0.001;
          End
          Ang=Ang+1;
    End
  End
  T2 = gettime();
  Time_Interval=T2 - T1; //unit is 1/100 sec
  Write(Time_Interval);
End
```

Fig. 5: Algorithm for Finding Time for Processing Using Library Function Call Method

```
//Processing time of sin function
//for N x 360 x 1000 calculations
for N = 1 to 29 step 2 do
Begin
  T1 = gettime();
  For i = 0 to N
  Begin
    Ang=0;
    while (Ang<360)
    Begin
          for FracAng=0 to 0.999 do
          Begin
            FracAngIndex=1000 x FracAng;
            Value =        COS[Ang] x
                   SinFrac[FracAngIndex] +
        SIN[Ang]x CosFrac[FracAngIndex];
FracAng+=0.001;
          End
          Ang=Ang+1;
    End
  End
  T2 = gettime();
  Time_Interval=T2 - T1; //unit is 1/100 sec
  Write(Time_Interval);
End

//Processing time of cos function
//for N x 360 x 1000 calculations
for N = 1 to 29 step 2 do
Begin
  T1 = gettime();
  For i = 0 to N
  Begin
    Ang=0;
    while (Ang<360)
    Begin
          for FracAng=0 to 0.999 do
          Begin
            FracAngIndex=1000 x FracAng;
            Value = COS[Ang] x
                   CosFrac[FracAngIndex] -
                   SIN[Ang] x
SinFrac[FracAngIndex];
            FracAng+=0.001;
          End
          Ang=Ang+1;
    End
  End
  T2 = gettime();
  Time_Interval=T2 - T1; //unit is 1/100 sec
  Write(Time_Interval);
End
```

Fig. 6: Algorithm for finding time for processing using Array Access method

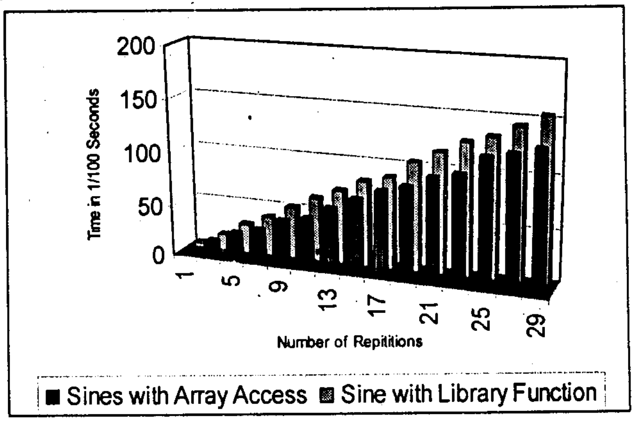

Fig. 7: Time analysis for computation of Sines

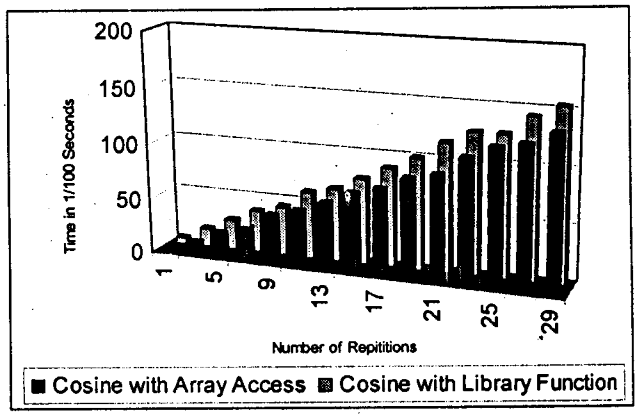

Fig. 8: Time analysis for computation of Cosines

Conclusion

The proposed method involves four accesses to the arrays while calculating value for one angle, two multiplications, and one addition or subtraction. These calculations are obviously less than the calculation of Maclaurin's Series but require processor time more than simple access of RAM. The method discussed in this paper is an effort to provide a general-purpose solution that can be adopted in variety of situations and avoid excessive use of storage space hence such compromise is necessary.

The method mentioned in this paper is considerably affected by the speed and architecture of the processor used. As high speed processors powered with math-coprocessors are developed, the time required to compute the series in the built-in library functions reduces. Hence the advantage of time with array access may be less visible on high-end processors as compared to processors used in personal computers. The time analysis provided in section 4.3 is implemented on an Intel 1.0 GHz processor which is one of the latest high-end processors hence the advantage in the processing time is apparently not very large.

ferences

arlie Morey, 3D Graphics for the World Wide Web : PROCESS OVERVIEW, http:/ /www. stars. com/ Authoring/ Graphics/3d/3d-demo2.html

Mubarak Shah, 2000. Fundamentals of Computer Vision, Class Notes for Computer Vision Lab, University of Central Florida, USA.

romi Yaguchi, Springer-Verlag, 1991. 3D CAD Principles and Applications, H. Toriya, H. Chiyokura, ISBN 0-387-56507-8.

ward Anton, 1995. Calculus and Analytical Geometry, John Wisely & Sons.

ren Heiny, 1994. Power Graphics Using Turbo C++, John Wiley & Sons Inc., ISBN 0-471-30929.

MSDN, Visual C++ Online Documentation.

Mathew Klare , 3D Studio Max : Pushing 3-D to the max, http:// www. znet.com /cshopper /content/ 9610/csh p0012.html

Muhammad Zaheer Aziz & Prof. Dr. Khalid Rashid 2000. Speeding Up Calculation of Sine and Cosine for Integer Degrees, Computer News .

Roger T. Stevens & Christopher D. Watkins, 1993. Advanced Graphics Programming in C & C++, BPB Publications.

Roger T. Stevens, 1993. Graphics Programming in C, BPB Publications.

Turbo C++ Online Documentation