Ditch the Decimal Data Type and Use Double in Your C# Code

John F. Gnazzo

Introduction

Currently there are four (4) native numeric data types in C#, including Integer, Float, Double and Decimal. This blog will discuss why using the Double data type over the Decimal datatype to significantly improve performance with almost no loss in computational accuracy.

Integer (Integral) Data Type

An **integral data type** [1]is used to store integer values. The following table shows the sizes and ranges of the integral types, which constitute a subset of simple types.

Type	Range	Size
sbyte	-128 to 127	Signed 8-bit integer
byte	0 to 255	Unsigned 8-bit integer
char	U+0000 to U+ffff	Unicode 16-bit character
short	-32,768 to 32,767	Signed 16-bit integer
ushort	0 to 65,535	Unsigned 16-bit integer
int	-2,147,483,648 to 2,147,483,647	Signed 32-bit integer
uint	0 to 4,294,967,295	Unsigned 32-bit integer
long	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	Signed 64-bit integer
ulong	0 to 18,446,744,073,709,551,615	Unsigned 64-bit integer

Float and Double Data Types

Float and Double data types are part of the **Floating Data Type**[2]. The following table shows the precision and approximate ranges for the floating-point types.

[1] Microsoft MSDN - https://msdn.microsoft.com/en-us/library/exx3b86w.aspx

Type	Approximate range	Precision
float	±1.5e−45 to ±3.4e38	7 digits
double	±5.0e−324 to ±1.7e308	15-16 digits

Decimal Data Type

The **Decimal** keyword indicates a 128-bit data type. Compared to floating-point types, the **Decimal** type [3]has more precision and a smaller range, which makes it appropriate for financial and monetary calculations. The approximate range and precision for the **decimal** type are shown in the following table.

Type	Approximate Range	Precision	.NET Framework type
Decimal	(-7.9 x 10^{28} to 7.9 x 10^{28}) / ($10^{0 \text{ to } 28}$)	28-29 significant digits	System.Decimal

Discussion

To test and assess both the accuracy and speed of using floats, double and decimals, a basic algorithm was selected that would compute the volume of a sphere. Since this is a calculation using Pi, which is NOT a whole number, the testing of the integral data type was discontinued.

The following figure shows the formula for computing the volume of a sphere.

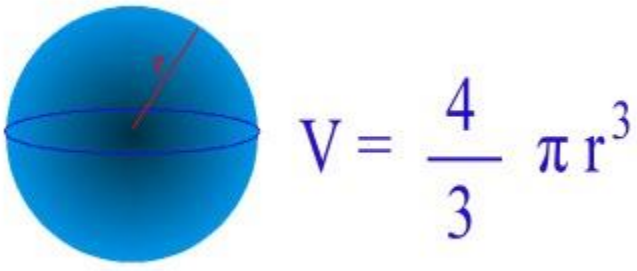

Figure 1 Volume of a Sphere

[2] Microsoft MSDN - https://msdn.microsoft.com/en-us/library/9ahet949.aspx
[3] Microsoft MSDN - https://msdn.microsoft.com/en-us/library/364x0z75.aspx

A software program was created to test both the accuracy and speed of the float, double and decimal data types. To get a good set of results, the calculation was iterated over a range of values and the test was repeated a number of times. The program is listed in the section below.

In this program, the calculation was conducted 1,000,000 times for each data type.

Program

```
        public List<string> Test(int iterations)
        {
            var results = new List<string>();

            float floatResult = 0.0f;
            double doubleResult = 0.0;
            decimal decimalresult = 0.0m;

            _stopWatch.Start();
            for (float i = 0, j = 0; i < iterations; i++, j += 0.000001f)
            {
                floatResult = floatResult + (j * j * j * floatPi) * 4.0f / 3.0f;
            }
            _stopWatch.Stop();

            var floatTime = _stopWatch.Elapsed.TotalMilliseconds;
            results.Add(item: $"Float  : {floatTime} {floatResult:N10}");

            _stopWatch.Reset();

            _stopWatch.Start();
            for (double i = 0, j = 0; i < iterations; i++, j += 0.000001)
            {
                doubleResult = doubleResult + (j * j * j * doublePi) * 4.0 / 3.0;
            }
            _stopWatch.Stop();

            var doubleTime = _stopWatch.Elapsed.TotalMilliseconds;
            results.Add(item: $"Double : {doubleTime} {doubleResult:N10}");

            _stopWatch.Reset();

            _stopWatch.Start();
            for (decimal i = 0, j = 0; i < iterations; i++, j += 0.000001m)
            {
                decimalresult = decimalresult + (j * j * j * decimalPi) * 4.0m / 3.0m;
            }
            _stopWatch.Stop();

            var decimalTime = _stopWatch.Elapsed.TotalMilliseconds;

            results.Add(item: $"Decimal: {decimalTime} {decimalresult:N10}");

            results.Add( $"Speed: {(floatTime/decimalTime)} {(doubleTime /
decimalTime)} ");
```

```
            results.Add($"Accuracy: {((decimalresult - (decimal) floatResult) /
decimalresult)} {((decimalresult - (decimal)doubleResult) / decimalresult)} ");

            return results;
        }
```

Results

The following table shows the relative results of the accuracy and speed of using Float, Double, and Decimal data types to compute the volume of a sphere over 1,000,000 iterations per the code above.

Data Type	Accuracy (percent)	Computation (msec)	Speed over Decimal
Float	+- 2.5	6.97	105.1 X Faster
Double	+- 0.000000000645	6.74	108.7 X Faster
Decimal	Baseline	732.58	Baseline

Based on the table above, the accuracy of the calculation is within 2.5 percent for Float and 0.000000000645 percent for Double. The speed increase over decimal is 105.1 and 108.7 (over 100 times faster) for Float and Double data types, respectively.

Conclusion

The performance of a Double data type is over 100 times faster and nearly as accurate as a Decimal. In cases where perfect accuracy is NOT required, one should definitely choose the Double data type.

Choosing to do calculations using Double data type will greatly improve application performance and help business applications handle greater volumes of users, resulting in less hardware requirements, less investment in capital resources and a better user experience.