# Geometry Library

An easy-to-implement function library providing numeric operations to handle solutions to typical problems encountered in the field of geometry, including Bezier Curves, Linear Interpolation (Lerp), and Trigonometry.

{Line}

**NEW! Simple 3D projection of lines and points!** Create a Camera3D or CameraOrtho, set its Position and LookAt properties, then feed it points and lines from world space all day long.

{Line}

## Table of Contents

You can jump to any section of this page from the following list.

* [25 Years of .NET](#_25_Years_of).
* [Yet Another Geometry Library](#_Yet_Another_Geometry).
* [Installation](#_Installation).
* [Usage Notes](#_Usage_Notes). Including a link to the full API documentation here on GitHub.

{Line}

## 25 Years of .NET

Although .NET doesn't officially turn 25 until February 13, 2027, I'm starting the celebration a little early.

To commemorate 25 years since the public release of the .NET framework, I'm open sourcing this and several other of my long-lived libraries and applications. Most of these have only previously been used privately in our own internal company productivity during the early 21st century but I hope they might find a number of new uses to complete the next 25 years.

I have every intention of keeping these libraries and applications maintained, so if you happen to run into anything you would like to see added, changed, or repaired, just let me know in the Issues section and I'll get it done for you as time permits.

{Line}

Sincerely,

**Daniel Patterson, MCSD (danielanywhere)**

{Line}

## Yet Another Geometry Library

Geometry is a surprisingly persistent companion in my work. Time and again, I find myself tackling scenarios where geometric functionality is not just helpful - it’s essential. Whether it’s displaying a line illustration projected in 3D, converting vectors to absolute coordinates, calculating the length of a guy wire with a Bézier-shaped sag, or determining the number of passes required for a mill bit to clear an area bounded by a polyline, geometry is always at the core of the solution. These repeated encounters inspired me to develop and maintain this geometric function library.

This library has a long history, originating in 1997 when I first created a version of it in the VBA scripting language to extend Microsoft Excel’s capabilities through their Office Automation extensions. It quickly became evident that its utility extended far beyond those early roots, prompting me to convert the library to C# in 2001. Since then, it has been an integral tool in my internal .NET projects, continuously evolving to meet new challenges.

{Line}

### Geometric Descriptors

In the current version, you will find these primitive geometric descriptor classes that can be used generically anywhere:

* **FArea**. Many of the same competencies as the more familiar RectangleF, but with support for negative area.
* **FColor4**. An RGBA color holder.
* **FEllipse**. A self-contained ellipse class with everything you need for handling an ellipse, including returning the area or perimeter, the bounding box of the shape, the focal points, the length of the imaginary string used to physically draw an edge around the shape in the real world, a coordinate on the edge at any angle, and intersections on a line.
* **FLine**. A floating point line.
* **FLine3**. A three-dimensional single precision floating point line.
* **FMatrix2**. A 2x2 linear matrix.
* **FMatrix3**. A 3x3 affine matrix.
* **FMatrix4**. A 4x4 affine matrix.
* **FPath**. Collection of single floating-point points.
* **FPoint**. A single precision floating-point coordinate.
* **FPoint3**. A three-dimensional single precision floating-point coordinate.
* **FQuaternion**. A quaternion rotation, complete with FromAxisAngle / ToAxisAngle, FromEuler / ToEuler (ZYX order), FromPitchRollYaw / ToPitchRollYaw (XYZ order), both Lerp and Slerp, Multiplication, Division, Dot Product and all of the other methods and functionality you would expect.
* **FRotation3**. A three-dimensional single precision floating-point rotation, similar to a point.
* **FScale**. A single floating-point scaling value.
* **FSize**. A single floating-point size.
* **FVector2**. A single floating-point 2-position array vector for use with matrix operations.
* **FVector3**. A single floating-point 3-position array vector for use with matrix operations.
* **FVector4**. A single precision floating-point4-position array vector for use with 4x4 matrix operations and other tasks.

Our geometric descriptors differ from others on two main features. First, all of our geometric descriptors are first-class objects, which allows you to pass everything by reference, maintain multiple references to the same coordinate, whose member values can be changed singularly from everywhere, and other benefits one gets from single instance storage. Secondly, many of our geometric descriptors provide event-based support for changing values and other notable circumstances.

{Line}

### Utility Classes

The Geometry library also contains the following utility classes for reaching answers in different categories.

* **Bezier Class**. A high-performance, lightweight module for identifying points along any type of Bézier curve. Whether your focus is precision or speed, this class delivers both, enabling quick and efficient calculations even for complex curves.
* **Camera3D**. A tiny, tiny, tiny 3D camera, performing all of its operations in raw trigonometric functions, making it capable of more than 25x CPU performance over current 3D systems. This renderer differs from the status quo in two major areas: 1) All vertices are assumed to be pre-positioned in their world locations and states, and 2) The camera is moved to its stated location to view the world as it is, as opposed to moving the entire world in front of a stationary camera, as is generally accepted practice. Together, this set of changes in philosophy reduce what would otherwise be a giant 3D library to just a handful of lines of code, as you can see inside. The catch on using it? Only individual 3D points and lines are rendered so far. You are invited to get involved if you would like to see this performance potential explode into the mainstream.
* **CameraOrtho**. An orthographic sibling to the tiny 3D camera, and nearly identical in operation, except for the projection stage. The main difference in preparation between the perspective Camera3D and the orthographic CameraOrtho is that in CameraOrtho, you specify a TargetObjectWidth, in host units, instead of a horizontal FieldOfView, in degrees.
* **Circle Class**. Provides intuitive tools and methods for analyzing and interacting with circles, including slice-based angles, bounding boxes, quadrant mapping, and spatial quadrant analysis.
* **GeometryUtil Class**. Generally, the 'Util' class of my libraries provide base singleton values and static support methods to all of the classes within the associated library, but I never mark these as internal in case they might be helpful to anyone else using the library.
* **Linear Class**. Focused on basic linear interpolation, this class includes overloads of the Lerp method, making it simple to interpolate scalar values or points with precision.
* **SlopeIntercept Class**. Although this class also utilizes linear math, this class provides dedicated support for lines using the *slope intercept* technique.
* **Trig Class**. Equipped with all the standard tools for working with angles and trigonometric functions, it offers seamless support for angular computations. Grandfathered in, this class also contains some methods that might be more suitable in the newer Circle class. Feedback on this would be welcome.

Through these features, this library aims to simplify and streamline geometric operations, empowering you to tackle projects with confidence and ease.

Other related classes from around our software environment will be added soon to this library, so stay tuned...

{Line}

## Installation

You can include this library in any .NET project using any supported programming language or target system. This library compiles as **.NET Standard 2.0** and is available in **NuGet** as

{Center}{Bold}{Big}Dan's Geometry Library{/Big}{/Bold}{/Center}

{Line}

**Instructions For Installation**

In **Visual Studio Community** or **Visual Studio Professional** editions:

* Right-click your project name in **Solution Explorer**.
* From the context menu, select **Manage NuGet Packages**.
* Click **Browse**.
* In the **Search** textbox, type **Dan's Geometry Library**.
* Accept the license agreement.
* In your code add the header line **using Geometry;**

{Line}

In **Visual Studio Code**:

* Run the command **NuGet: Add NuGet Package** (typically [Ctrl][Shift][P]).
* If there are multiple projects in the solution, select the open project to which the package will be assigned.
* In the **Search** textbox, type **Dan's Geometry Library**.
* Select the package.
* Select the version you wish to apply.

{Line}

## Usage Notes

This library is intended to be used on any target system, avoiding any kind of Windows dependencies whatsoever. As a result, replacements for GDI+ dimensional objects, like Point, Rectangle, and others normally found in .NET system libraries like System.Drawing have been defined for generic public use with or without a statically typed graphics system.

To see working examples of various uses of this library, see the **Source/GeometryExample** folder, where I add various tests and use-cases to a stand-alone application before publishing each version.

If you would like to see a bigger-picture view of the library in daily use, review some of the source of my other GitHub project **danielanywhere/ShopTools**. That project uses Dan's Geometry Library to draw graphics, calculate distances, and perform a lot of the heavy lifting.

For the full documentation of this library, you can access the API in HTML format at the associated GitHub user page library <https://danielanywhere.github.io/Geometry>.