# WPF Surface Plot 3D Documentation

***Release 0.1.0***

**Brittany Welsh**

February 08, 2016

CHAPTER 1

# Easy, attractive 3D plots in WPF

WPF-SurfacePlot3D is a tiny C# library containing easy-to-use 3D surface plotting components for WPF (.NET) applications.

You can get started in as few as four lines:

```
var myPlot = new SurfacePlotViewModel();
myPlotView.DataContext = myPlot;
function = (x, y) => x * y;
viewModel.PlotFunction(function, -1, 1);
```

CHAPTER 2

# Features

- Flexible data input formats:
  - Plot functions directly
  - Use your own data arrays
- Easy data binding for real-time updates
- Leverage the beauty and power of Helix 3D Toolkit

CHAPTER 3

# Installation

While the project is in beta, simply grab the class files from the Github repo and add them to your own project. For more information, read the *Installation* section.

COMING SOON: Installation via NuGet package manager (planned for version 1.0.0).

CHAPTER 4

# Contribute

This project is organized using Github.

- Please report all bugs, feature requests, and other issues through the issue tracker.
- Feel free to add your thoughts to the project wiki!

CHAPTER 5

# Contact

If you're having issues or just want to send along some feedback, feel free to email me: hello@brittanywelsh.com.

CHAPTER 6

# License

This project is licensed under the GNU General Public License (version 3).

## 6.1 Installation

### 6.1.1 Download the demo project

If you'd just like to test out the library, this option is for you!

Simply grab the repo in your preferred format (zip, git clone, etc) and open up the file `WPFSurfacePlot3D/WPFSurfacePlot3D.sln`.

At this point, ensure that HelixToolkit.Wpf and Extended WPF Toolkit are installed (I recommend installing via NuGet).

From there, you should be able to build the project and play around with a few different surface plots. Enjoy!

### 6.1.2 Import SurfacePlot3D files into your existing project

If you just want to use the SurfacePlot3D files, the easiest way to import them to your existing project is just to copy the following four files into your project:

- SurfacePlotModel.cs
- SurfacePlotView.xaml
- SurfacePlotView.xaml.cs
- SurfacePlotVisual3D.cs

They can be accessed via the namespace `WPFSurfacePlot3D`. You will also have to ensure that HelixToolkit.Wpf is installed in your project.

### 6.1.3 Coming soon

Installation via NuGet package manager is planned for an upcoming release. Stay tuned!

## 6.2 Tutorial

So you've got the *files you need* located somewhere in your demo project, right?

To start using 3D plots, we need to follow a simple set of steps.

In the code-behind for the view (Window, App, etc.) where you'd like to display a surface plot, first ensure that the appropriate namespace is referenced with a `using` statement:

```
using WPFSurfacePlot3D;
```

You'll also have to ensure the namespace is called from the XAML file as well:

```
<Window x:Class="MyWindow" xmlns:SurfacePlot="clr-namespace:WPFSurfacePlot3D">
```

Next, go to the XAML file. You can grab a SurfacePlotView control from your toolbox and drag it into your XAML file, or simply add the following XAML:

```
<SurfacePlot:SurfacePlotView  x:Name="mySurfacePlotView" />
```

Once you've added mySurfacePlot, now you need to set up the DataContext in the code-behind. Back in your .xaml.cs file, add the following:

```
mySurfacePlotModel = new SurfacePlotModel();
mySurfacePlotView.DataContext = mySurfacePlotModel;
```

You can, of course, rename `mySurfacePlotModel` and `mySurfacePlotView` to whatever you please.

---

**Note:** After you set up your data binding by setting the DataContext, the object called mySurfacePlotModel is now your main point of control to interact with the plot.

---

Great - now you're ready to start *plotting functions*!

## 6.3 Plotting Functions

Let's add our own plot based on a mathematical function. First we have to define a function that has two "input" variables and one "output variable", where all of these variables are a `double` type - hence, we will define a function of the type `Func<double, double, double>`. (Note: you have to include `using System;` to get access to this data type.)

One simple way of doing this is defining our function via the lambda operator:

```
Func<double, double, double> sampleFunction = (x, y) => x * y;
```

Easy enough, right? let's define something even more interesting:

```
sampleFunction = (x, y) => 10 * Math.Sin(Math.Sqrt(x * x + y * y)) / Math.Sqrt(x * x + y * y);
```

Now we're ready to plot our function. It's as easy as passing in the function to the appropriate method:

```
PlotFunction(mySampleFunction);
```

This will plot your function in the range x:[-1, 1], y:[-1, 1]. If you want to change the default range, you can do so through the same function:

```
PlotFunction(mySampleFunction, -10, 5);
```

This plots the function in the range x:[-10, 5], y:[-10, 5].

If you want to have even more control over the range, you can do that too! Let's plot the function in the domain x:[-3, 9], y:[-20, 0]:

```
PlotFunction(mySampleFunction, -3, 9, -20, 20);
```

That's everything you need to get started with your plot! From here, why not customize your plot's appearance? Or you can learn how to plot data that you define yourself.

Finally, it's worth noting that in order to plot the function, we are first "sampling" that function over the defined domain. The default sample size in both x and y directions is 100 points, resulting in a collection of point objects of size 100x100 = 10000. You can add arguments to the PlotFunction method in order to change the sample size.

## 6.4 Plotting 3D Data

This is where SurfacePlot3D really shines: providing an easy entry point for your custom 3D data set. Depending on the type of data you're starting with, there are a number of different arguments you can pass into the PlotData function.

### 6.4.1 Plotting a 2D Array of Values

This is the easiest possible case. Let's say we've got a 2D array of doubles set up:

```
double[,] arrayOfPoints = new double[10, 15];
for (int i = 0; i < 10; i++)
{
  for (int j = 0; j < 15; j++)
  {
    arrayOfPoints[i, j] = Math.Cos(Math.Abs(i) + Math.Abs(j)) * (Math.Abs(i) + Math.Abs(j));
  }
}
```

Now all we have to do is plug it into our PlotData function:

```
PlotData(arrayOfPoints);
```

Great! But you should notice how only passing in an array of points will give you automatic indexing: that is, your x and y axes will be indexed from 0 to n-1, where n is the number of points in either dimension.

If you have a function, like say you wanted to plot the same function as above, but you wanted to plot 20 and 25 points between 0 and 1 in each the x and y axes, respectively. For this, we also need to define a double array of the corresponding size for each axis. We might set up such a plot like so:

```
int N = 20;
int M = 25;

double[] xArray = double[N];
double[] yArray = double[M];
double[,] zArray = double[N, M]; // Notice how the size of the array in each dimension has to match u
for (int i = 0; i < N; i++)
{
  double x = i / N;
  for (int j = 0; j < M; j++)
  {
```

```
    double y = j / M;
    arrayOfPoints[i, j] = Math.Cos(Math.Abs(x) + Math.Abs(y)) * (Math.Abs(x) + Math.Abs(y));
  }
}
```

Finally, we'll plot each of the arrays:

```
PlotData(zArray, xArray, yArray);
```

Perfect!

### 6.4.2 Plotting an Array of 3DPoint Objects

Since the SurfacePlot3D library works primary through Point3D arrays, this is the most straightforward plot. You can plot a 2-dimensional array of Point3D objects by simply passing the array into the PlotData function:

```
PlotData(point3DArray);
```

## 6.5 Editing Plot Appearance

So you have a plot on the screen - that's great! But it might not look exactly the way you want it to. This section outlines some different options you can use to tweak the appearance of your plot.


## 6.6 SurfacePlotModel

[SurfacePlotViewModel API] - complete list of functions you can call on SurfacePlotModel objects.

### 6.6.1 File Organization/Class Relationships

The general design behind these classes is straightforward.

The view inherits from UserControl. This allows the designer to easily grab and manipulate the control in the XAML editor of their application.

The View exposes various DependencyProperties. In general, the View is concerned just with plotting. The Model, on the other hand, takes care of anything needed to get from user input to the plotting data. This usercontrol basically takes its datacontext from the Model and creates the View needed (in a Helix Viewport), interacting with the methods from SurfaePlotVisual3D when necessary.

SurfaePlotVisual3D inherits from Visual3D and it is the collection of 3D visual objects that are drawn to the screen.

The Model is intended to be the main point of entry for the user; everything else is intended to be dealt with privately.