

LightningChart®

User's Manual

About this document

This document is a brief User's Manual, reference of **Arction LightningChart Ultimate .NET**. Only essential key features are explained. Hundreds of classes, properties or methods are not described in this document. Run the provided demo applications to get a quick preview of some LightningChart features. The source code of included demo applications helps understand how to use LightningChart components in code.

All code examples in this document are written in C# language. Majority of demo applications provide code preview in C# and Visual Basic .NET, as well.

And remember, don't hesitate to contact support (support@arction.com) if you have any question!

Applies to LightningChart .NET, v.8.4.1



Copyright Arction Ltd 2009-2019. All rights reserved.

LightningChart is registered trademark by Arction Ltd.

www.arction.com

www.lightningchart.com

Contents

1. Overview.....	17
1.1 Chart editions	17
1.2 Components	18
1.3 Namespaces.....	19
2. Installation.....	20
2.1 Before installing.....	20
2.2 Run the setup wizard.....	20
2.3 Adding Arction components manually to Visual Studio Toolbox.....	20
2.4 Configuring Visual Studio 2010-2017 help manually.....	21
2.4.1 Visual Studio 2010	21
2.4.2 Visual Studio 2012-2017.....	22
2.5 Code parameters and tips by Visual Studio IntelliSense	23
2.6 Selecting target framework.....	23
3. License management.....	25
3.1 Adding license.....	25
3.2 Removing a license	26
3.3 Extracting Deployment Key	27
3.4 Applying Deployment Key in an application.....	27
3.5 Running with Deployment Key on development computer.....	29
3.6 Running with debugger	29
3.7 Trial period	29
3.8 Floating licenses	29
4. LightningChart Ultimate component.....	31
4.1 Adding from toolbox into Windows Forms project.....	31
4.1.1 Properties	31
4.1.2 Event handlers.....	31
4.1.3 Best practices concerning version updates	32
4.2 Adding from toolbox into WPF project	32
4.2.1 Properties	32
4.2.2 Event handlers.....	33
4.3 Adding into Blend WPF project	33
4.3.1 Best practices concerning version updates	34
4.3.2 Preventing blurring of the chart	34
4.4 Object model	35
4.4.1 Differences between Windows forms and WPF.....	36
4.5 LightningChart Views.....	36

4.6	View and zooming area definitions	36
4.7	Setting background fill.....	38
4.7.1	Setting transparent background.....	40
4.8	Configuring appearance / performance settings	41
4.9	DPI handling.....	44
4.9.1	DpiHelper class	44
5.	ViewXY	45
5.1	Axis layout options	48
5.1.1	Setting how axes are placed	48
5.1.1.1	X axis automatic placement.....	48
5.1.1.2	Y axis automatic placement.....	50
5.1.2	Graph segments and Y axes placement in them	53
5.1.2.1	Layered	53
5.1.2.2	Stacked	53
5.1.2.3	Segmented.....	54
5.1.3	Axis grid strips.....	55
5.1.4	Other AxisLayout options	56
5.2	Y axes	57
5.2.1	AxisY class properties	57
5.2.2	Tick value labels formatting.....	57
5.2.3	Value type.....	58
5.2.4	Range setting	59
5.2.5	Restoring range.....	59
5.2.6	Divisions.....	59
5.2.7	Grid	59
5.2.8	Custom ticks	60
5.2.9	Reversed X and Y axis	61
5.2.10	Logarithmic axes	61
5.2.10.1	Exponential presentation for 10 base	61
5.2.10.2	Natural logarithm	62
5.2.11	Converting between axis values and screen coordinates	62
5.2.12	MiniScale	63
5.3	X axis	63
5.3.1	Real-time monitoring scrolling	63

5.3.1.1	None	63
5.3.1.2	Stepping.....	64
5.3.1.3	Scrolling	64
5.3.1.4	Sweeping	66
5.3.1.5	Triggering.....	66
5.3.2	Scale breaks	67
5.4	Margins.....	70
5.5	ViewXY series, general.....	71
5.6	Point line series	71
5.6.1	Line style	72
5.6.2	Points style.....	72
5.6.3	Coloring points individually	72
5.6.4	Adding points.....	73
5.6.5	Adding points, alternative way.....	73
5.7	Sample data series.....	74
5.7.1	Y precision.....	75
5.7.2	Adding points.....	75
5.8	Freeform point line series	76
5.9	Advanced line coloring of line series.....	77
5.9.1	Y-value based coloring of line and fill with value-range palette	77
5.9.2	Custom shaping and coloring with CustomLinePointColoringAndShaping event.....	78
5.10	High-low series	80
5.10.1	Fill, line and point styles	80
5.10.2	Limits.....	81
5.10.3	Coloring by value-range palette	82
5.10.4	Adding data.....	82
5.11	Area series	83
5.11.1	Adding data.....	83
5.12	Bars.....	85
5.13	Stock series	87
5.13.1	Setting data to StockSeries	89
5.13.2	Setting X axis to date display	89
5.13.3	Custom formatting of appearance	90
5.13.4	Applying Scale breaks	90

5.14	PolygonSeries	91
5.14.1	Setting data to a Polygon.....	91
5.14.2	Enabling complex / intersecting fills.....	92
5.15	LineCollections.....	92
5.15.1	Setting data to a LineCollection.....	93
5.16	IntensityGridSeries	94
5.16.1	Setting intensity grid data	96
5.16.2	Creating intensity grid data from bitmap file	97
5.16.3	Fill styles	98
5.16.4	Rendering as pixel map.....	98
5.16.5	ValueRangePalette	99
5.16.6	Wireframe.....	100
5.16.7	Contour lines	100
5.16.8	Contour line labels.....	102
5.17	IntensityMeshSeries	102
5.17.1	Setting intensity mesh data, when geometry changes	104
5.17.2	Setting intensity mesh data, when geometry does not change	104
5.17.2.1	Creating the series and its geometry.....	105
5.17.2.2	Updating the values periodically	105
5.18	Bands	105
5.19	Constant lines	106
5.20	Annotations	107
5.20.1	Controlling target and location.....	108
5.20.2	Using mouse to move, rotate and resize.....	109
5.20.3	Adjusting appearance	110
5.20.4	Size settings	110
5.20.5	Keeping text area visible.....	110
5.20.6	Displaying annotation over axes.....	110
5.20.7	Clipping inside graph	111
5.20.8	Controlling the Z order	111
5.20.9	LayerGrouping performance optimization	111
5.20.10	Converting between axis values and screen coordinates	112
5.21	Legend box.....	113
5.21.1	Hiding / showing a series from legend box	114

5.21.2	Preventing a series from listing itself in the legend box	114
5.21.3	Selecting in which legend box to show a specific series	114
5.21.4	Selecting in which graph segment to show a legend box	114
5.21.5	Hiding check boxes	114
5.21.6	Hiding icons	114
5.21.7	Hiding intensity series palette scales.....	115
5.21.8	Controlling positions.....	115
5.21.9	Allocating space for legend boxes between graph segments	116
5.21.10	Alignment of legend boxes in segment gap	117
5.21.11	Horizontal alignment of several legend boxes sharing the same margin	117
5.21.12	Resizing and moving legend boxes	118
5.22	Zooming and panning	119
5.22.1	Zooming with touch screen	120
5.22.2	Panning with touch screen	120
5.22.3	Left mouse button action	120
5.22.4	Right mouse button action	120
5.22.5	RightToLeftZoomAction.....	121
5.22.6	Zooming with mouse button	121
5.22.6.1	Zoom in/out by clicking	121
5.22.6.2	Zoom in with rectangle.....	122
5.22.6.3	Configuring zoom out rectangle	122
5.22.7	Zooming with mouse wheel	122
5.22.8	Zooming and panning with mouse wheel, over axis	122
5.22.9	Panning with mouse button	123
5.22.10	Enabling/disabling Ctrl, Shift and Alt.....	123
5.22.11	Zoom in/out with code	123
5.22.12	Zooming an axis by code	123
5.22.13	Rectangle zooming about a configurable origin.....	123
5.22.14	Linking Y axes zoom with same units	124
5.22.15	Automatic Y fit	125
5.22.16	Aspect ratio.....	125
5.22.17	Excluding specific X or Y axes from zooming and panning operations.....	126
5.23	DataBreaking by NaN or other value.....	126

5.24	ClipAreas.....	128
5.25	Maps	130
5.26	Vector maps.....	131
5.26.1	Selecting active map.....	131
5.26.2	Aspect ratio.....	132
5.26.3	Layers and their appearance settings.....	133
5.25.3.1	Setting individual fill and border style for each layer item	134
5.26.4	Mouse interactivity.....	135
5.26.5	Background photos.....	136
5.26.6	Combining other series with maps.....	137
5.26.7	Importing maps from ESRI shape file data	139
5.26.7.1	Programming interface for importing shp data	139
5.26.7.2	Dialogs	140
5.26.7.2.1	Shapefile Selection Dialog	140
5.26.7.2.2	Select Record Encoding and Invalid Name Fields	141
5.26.7.2.3	Layer data selection dialog.....	142
5.26.7.2.4	Item filter	144
5.26.8	Importing and replacing map layers.....	144
5.27	Tile maps.....	146
5.27.1	HERE.....	147
5.28	Stencil areas.....	148
5.29	Line series cursors	150
5.29.1	Solving the data values in the position of LineSeriesCursor.....	151
5.29.1.1	Accurate method, solving Y value by X value using data points array	151
5.29.1.2	Coarse method, solving Y screen coordinate by X coordinate using data points array	152
5.30	Event markers.....	153
5.30.1	Chart event markers	154
5.30.2	Line series event markers	154
5.31	Persistent series rendering layers	156
5.31.1	Creating the layer	157
5.31.2	Clearing the layer.....	158
5.31.3	Adjusting layer alpha	158
5.31.4	Rendering data into the layer.....	158
5.31.5	Disposing the layer	159

5.31.6	Anti-aliasing data in the layer	159
5.31.7	Getting list of layers.....	159
5.31.8	Some layer limitations to be aware of.....	159
5.32	Persistent series rendering intensity layers	160
5.32.1	Creating the layer	161
5.32.2	Clearing the layer.....	161
5.32.3	Changing palette colors	161
5.32.4	Adjusting the intensity effect of new trace and decay of old traces.....	161
5.32.5	Rendering data into the layer	161
5.32.6	Disposing the layer	162
5.32.7	Anti-aliasing data in the layer.....	162
5.32.8	Getting list of layers.....	162
6.	View3D.....	163
6.1	3D model and dimensions	164
6.1.1	World coordinates.....	164
6.2	Walls	165
6.3	FrameBox.....	166
6.4	Camera.....	166
6.4.1	Predefined cameras.....	169
6.4.2	Camera orientation mode	169
6.5	Lights.....	169
6.5.1	Directional light	170
6.5.2	Point of light	170
6.5.3	Lights and materials.....	170
6.5.4	Predefined lighting schemes	171
6.6	Axes	171
6.6.1	Location	172
6.6.2	Orientation	173
6.6.3	CornerAlignment	173
6.7	Margins.....	174
6.8	3D series, general	174
6.9	PointLineSeries3D.....	175
6.9.1	Point styles.....	175
6.9.2	Line styles	177

6.9.3	Adding points.....	177
6.9.3.1	Points.....	178
6.9.3.2	PointsCompact.....	178
6.9.3.3	PointsCompactColored.....	179
6.9.4	Coloring points individually.....	179
6.9.5	Setting points sizes individually.....	180
6.9.6	Multi-coloring line.....	181
6.9.7	Displaying millions of scatter points.....	181
6.10	SurfaceGridSeries3D.....	183
6.10.1	Setting surface grid data.....	184
6.10.2	Creating surface from bitmap file.....	185
6.10.3	Fill styles.....	185
6.10.4	Contour palette.....	187
6.10.5	Wireframe mesh.....	188
6.10.5.1	Some notes when using wireframe simultaneously with fill.....	190
6.10.6	Contour lines.....	191
6.10.7	Fadeaway.....	192
6.10.8	Scrolling surface data.....	192
6.11	SurfaceMeshSeries3D.....	194
6.11.1	Setting surface mesh data.....	195
6.12	WaterfallSeries3D.....	196
6.13	BarSeries3D.....	197
6.13.1	Bars grouping.....	197
6.13.2	Bar styles.....	200
6.13.3	Setting bar series data.....	201
6.13.4	Showing bars horizontally.....	202
6.14	MeshModels.....	203
6.14.1	Loading a model.....	204
6.14.2	Positioning, scaling and rotating the model.....	204
6.14.3	Enabling fill and wireframe.....	204
6.14.4	Custom-coloring fill.....	205
6.14.5	Custom-coloring wireframe.....	206
6.14.6	Reverse vertices winding order.....	206
6.14.7	Constructing MeshModel programmatically from vertices.....	206

6.14.7.1	Updating the bitmap fill efficiently.....	207
6.14.8	Tracing the model with mouse.....	208
6.15	VolumeModels	209
6.15.1	Loading data	209
6.15.2	Properties	209
6.15.3	Ray Function	211
6.15.4	Threshold.....	213
6.15.5	Slice Range.....	214
6.15.6	Sampling Rate Options	215
6.15.7	Smoothness	216
6.15.8	EmptySpaceSkipping.....	217
6.15.9	Opacity.....	218
6.15.10	Brightness and Darkness	218
6.16	Rectangle3D objects	219
6.17	Polygon3D objects.....	220
6.18	Zooming, panning and rotating	223
6.18.1	Mouse wheel zooming	223
6.18.2	Box zooming	224
6.18.3	ZoomPadding.....	224
6.18.4	ZoomToDataAndLabels.....	225
6.18.5	Rotating and panning	226
6.18.6	Zooming with touch screen	227
6.18.7	Panning with touch screen	227
6.18.8	Using mouse wheel over an axis	227
6.18.9	Zooming, rotating and panning by code.....	227
6.19	Legend boxes	228
6.19.1	Hiding surface series palette scales.....	228
6.19.2	Positioning legend boxes in View3D.....	229
6.20	Clipping objects within axis ranges.....	230
6.21	Annotation3D	231
7.	Coordinate system converters	232
7.1	SphericalCartesian3D.....	232
7.1.1	Converting from spherical to cartesian	233
7.1.2	Converting from cartesian to spherical	233

7.2	CylindricalCartesian3D.....	234
7.2.1	Converting from cylindrical to cartesian	235
7.2.2	Converting from cartesian to cylindrical	235
8.	ViewPie3D.....	236
8.1	Properties	237
8.2	Pie slices.....	237
8.3	Setting data by code	238
8.4	Viewing pie chart in 2D.....	239
9.	ViewPolar.....	240
9.1	Axes	241
9.1.1	Reversed axes	242
9.1.2	Setting rotation angles of the scales	243
9.1.3	Setting divisions.....	244
9.2	Margins.....	244
9.3	Legend boxes	246
9.3.1	Hiding palette scales.....	246
9.3.2	Legend box positioning in ViewPolar	247
9.4	PointLineSeries	248
9.4.1	Setting data.....	248
9.4.2	Palette coloring.....	249
9.4.3	Custom shaping and coloring with CustomLinePointColoringAndShaping event.....	249
9.5	AreaSeries.....	250
9.5.1	Setting data.....	250
9.6	Sectors	251
9.7	Annotations	251
9.8	Markers.....	252
9.9	Zooming and panning.....	252
9.9.1	Zooming operations and methods	253
10.	ViewSmith.....	255
10.1	Axis.....	255
10.2	Margins.....	259
10.3	Legend boxes	260
10.4	PointLineSeries	260
10.5	Setting data.....	261
10.6	Annotations	261
10.7	Markers.....	262

10.8	Zooming and panning	262
11.	Setting color theme	263
12.	Scrollbars	264
13.	Export and printing	265
13.1.1	Bitmap image export	265
13.1.2	Vector image export	265
13.1.3	Copy to clipboard.....	265
13.1.4	Capturing to byte array	266
13.1.5	Setting output stream for continuous frame writing	266
13.1.6	Printing	267
14.	LightningChart performance	268
14.1	Selecting the correct API edition	268
14.2	Set the rendering options correctly.....	268
14.3	Updating chart data or properties.....	268
14.4	Line series tips	270
14.5	Intensity series tips.....	270
14.6	3D Orthographic view tips.....	270
14.7	3D surface series tips.....	271
14.8	Maps tips	271
14.9	Hardware	271
15.	LightningChart notifications, error and exception handling	272
16.	ChartManager component	273
16.1	Chart interoperation, drag-drop.....	273
16.2	Memory management enhancement	273
17.	SignalGenerator component	274
17.1	Sampling frequency, Output interval and Factor	274
17.2	Sine waveforms	275
17.3	Square waveforms.....	276
17.4	Triangle waveforms	276
17.5	Noise waveforms	277
17.6	Frequency sweeps	277
17.7	Amplitude sweeps	278
17.8	Starting and stopping	278
17.9	Multi-channel generator with master-slave configuration	278
17.10	Output data stream	279
18.	SignalReader component	280
18.1	Key properties	280
18.2	Opening file quickly for playback	280

19. AudiolInput component	282
19.1 Properties	282
19.2 Methods	282
19.3 Events	283
19.4 Usage (WinForms)	283
19.4.1 Creation	283
19.4.2 Event handling	283
19.4.3 Configuring	284
19.4.4 Starting	284
19.4.5 Stopping	285
19.5 Usage (WPF)	285
19.5.1 Creation	285
20. AudioOutput component	286
20.1 Properties	286
21. SpectrumCalculator component	287
22. Headless mode	289
22.1.1 Headless Rendering	289
22.1.1.1 Additional initialization options	289
22.1.1.2 Capturing images	290
22.1.2 Limitations and Requirements	291
22.1.2.1 Threads	291
22.1.2.2 Chart Update	291
22.1.2.3 Engine support	291
22.1.2.4 Licensing	291
22.1.3 Example solution	292
23. Using Windows Forms chart in WPF application	294
23.1 How about using Arction Windows Forms controls in WPF?	294
23.2 Should I use Arction.WinForms.LightningChartUltimate in WPF?	294
24. Using LightningChart in C++ applications	297
24.1 Install required C++/CLR packages	297
24.2 Setting Visual Studio project	298
24.3 Creating LightningChart application in C++ project	300
25. Dispose pattern	303
25.1 Chart disposing	303
25.2 Disposing objects	303
26. Object model notes	304
26.1 Sharing objects between other objects	304

27. Deployment / distribution of LightningChart assemblies.....	306
27.1 Referenced assemblies	306
27.2 License key.....	307
27.3 Obfuscating application code	307
27.4 Obfuscating LightningChart code	307
27.5 XML files of Arction assemblies	307
28. Troubleshooting	308
28.1 Updating from older version	308
28.2 Web support.....	309
28.3 Running in Virtual Machine platforms	309
29. Credits.....	310
29.1 Intel Math Kernel library	310
29.2 Open-source projects	310

1. Overview

LightningChart Ultimate SDK is an add-on to Microsoft Visual Studio, consisting of data visualization related software components and tool classes for *WPF (Windows Presentation Foundation) and Windows Forms .NET* platforms.

Arction components are delivered for serious scientific, engineering, measurement and trading solutions, execution performance and very advanced features in special focus.

LightningChart components use low-level DirectX11 and DirectX9 GPU acceleration instead of slower GDI/GDI+ or WPF Graphics APIs. LightningChart has fallback to DirectX11/DirectX10 WARP software rendering when GPU is not accessible, such as in some virtual machine platforms.

1.1 Chart editions

For WPF, LightningChart component is available in different binding level editions, to balance between different performance and MVVM (Model - View - View-Model) bindability needs.

Chart edition	Properties binding	Series data binding	Per-data-point binding	Performance
<i>WPF (non-bindable)</i>	No	No	No	Excellent
<i>WPF (semi-bindable)</i>	Yes	Yes	No	Very good
<i>WPF (bindable)</i>	Yes	Yes	Yes	Good
<i>WinForms</i>	No	No	No	Best

Table 1-1. Bindability and performance matrix.

As a general starting point, Arction recommends Semi-bindable API.

- For best performance in WPF and multithreading benefits, select non-bindable chart.
- For good tradeoff between WPF bindability and performance, select semi-bindable chart.
- For full WPF MVVM design pattern support, select fully bindable chart.

Semi-bindable chart API is very similar to LightningChart v.6's WPF chart but comes with extended properties binding which also covers objects created in collections.

Different chart editions can be used in the same application. It's possible to make basic charts with fully bindable chart and bind the data while using the non-bindable chart for performance-critical tasks.

The collection properties of semi-bindable and bindable charts (such as ViewXY axes, 3D lights) are empty by default which supports XAML editor in full. In Non-bindable and WinForms collections are prefilled with default items.

Note! Non-Bindable WPF chart is not intended to be configured in XAML at all. Use it in code-behind.

1.2 Components

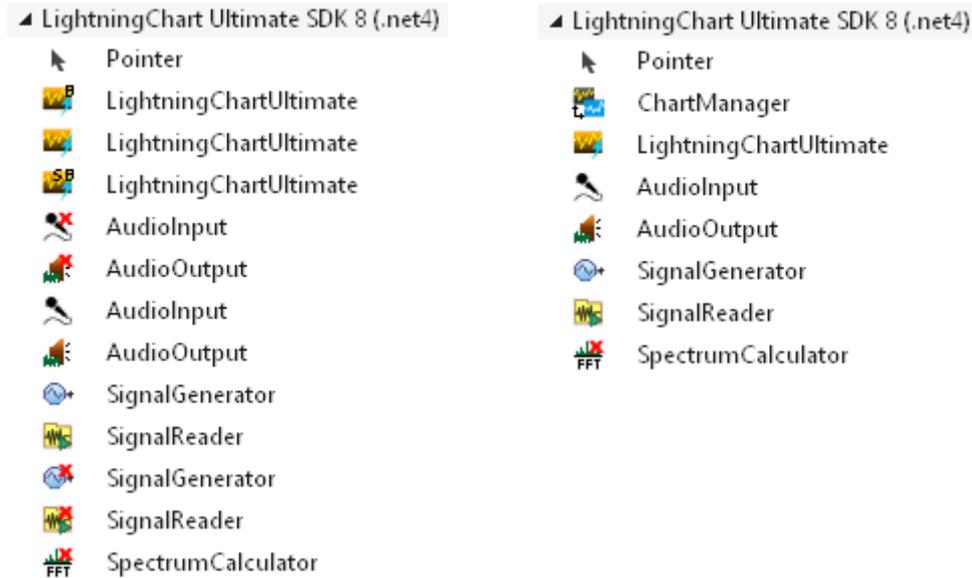


Figure 1-1. On the left, WPF toolbox components. On the right, WinForms toolbox components

Charting assembly

 **LightningChart Ultimate** The chart component. Visualizes data in various presentations.

In top corner of the icon, SB = Semi-bindable WPF chart and B = Bindable WPF chart

 **ChartManager** Controls interoperation of multiple charts components and real-time measurement memory management. See chapter 16.

SignalTools assembly

Components that don't have an UI, are marked with **X**.

 **AudiolInput** Reads waveform audio stream from a sound device. Line-in or microphone-in connectors are typical options available in a sound device. The real-time stream can be forwarded to other controls. See chapter 19.

 **AudioOutput** Plays back real-time data stream through the sound device, to speakers or line-output for example. Doesn't have to be an audio stream, any sampled real-time signal can be used. See chapter 20.



SignalGenerator Generates signal from multiple configurable waveform components. See chapter 17.



SignalReader Reads waveform data from a signal file, such as PCM formatted WAV. See chapter 18.



SpectrumCalculator Converts signal data (time domain) to spectrum (frequency domain), by using FFT (Fast Fourier Transform). Also contains methods for backwards conversion, frequency domain to time domain. See chapter 21.

1.3 Namespaces

Chart edition	Assembly name	Namespace root	XML namespace
WPF (non-bindable)	Arction.Wpf.Charting. LightningChartUltimate.dll	Arction.Wpf. Charting	<code>xmlns:lcunb="http://schemas.arction.com/charting/ultimate/"</code>
WPF (semi-bindable)	Arction. Wpf.SemibindableCharting. LightningChartUltimate.dll	Arction.Wpf. SemibindableCharting	<code>xmlns:lcusb="http://schemas.arction.com/semibindablecharting/ultimate/"</code>
WPF (fully bindable)	Arction. Wpf.BindableCharting. LightningChartUltimate.dll	Arction.Wpf. BindableCharting	<code>xmlns:lcufb="http://schemas.arction.com/bindablecharting/ultimate/"</code>
WinForms	Arction. WinForms.Charting. LightningChartUltimate.dll	Arction.WinForms. Charting	N/A

Table 1-2. Assembly names and namespaces of all LightningChart Ultimate editions.

2. Installation

2.1 Before installing

Check if the computer configuration meets the requirements

- DirectX 9.0c (shader model 3) level graphics adapter or newer, or DirectX11 compatible operating system for rendering without graphics hardware. DirectX11 compatible graphics hardware recommended.
- Windows Vista, 7, 8 or 10, as 32 bit or 64 bit, Windows Server 2008 R2 or higher
- Visual Studio 2010-2017 for development, not required for deployment
- .NET framework v. 4.0 or newer installed

2.2 Run the setup wizard

Right-click on the **LightningChart Ultimate SDK v8.exe**. The setup will install the components into Visual Studio toolbox. It also installs the help files associated with the toolbox controls. If components or help install fails, install them manually as instructed in the following sections.

When trialing LightningChart, **SetupDownloader.exe** is most likely used. This downloads and installs the SDK, meaning running LightningChart Ultimate SDK v8.exe explicitly is not required.

2.3 Adding Arction components manually to Visual Studio Toolbox

WinForms

1. Open Visual studio. Create a new **WinForms** project. Right-click on Toolbox, select **Add Tab** and give name "Arction"
2. Right-click on Arction tab, and select **Choose items...**
3. In **Choose Toolbox items** window, Select **.NET Framework Components** page. Click **Browse...**

Browse **Arction.WinForms.Charting.LightningChartUltimate.dll** and **Arction.WinForms.SignalProcessing.SignalTools.dll**, from the folder the components were installed on, typically **C:\Program Files (x86)\Arction\LightningChart Ultimate SDK v.8\LibNet4**, and click open. The components can now be found in the toolbox.

WPF

1. Open Visual studio. Create a new **WPF** project. Right-click on Toolbox, select **Add Tab** and give name "Arction"
2. Right-click on Arction tab, and select **Choose items...**
3. In **Choose Toolbox items** window, Select **WPF Components** page. Click **Browse...**

Browse **Arction.Wpf.Charting.LightningChartUltimate.dll**, **Arction.Wpf.SemibindableCharting.LightningChartUltimate.dll**, **Arction.Wpf.BindableCharting.LightningChartUltimate.dll** and **Arction.Wpf.SignalProcessing.SignalTools.dll**, from the folder the components were installed to, typically **C:\Program Files (x86)\Arction\LightningChart Ultimate SDK v.8\LibNet4**, and click open. The components can be now found in the toolbox.

2.4 Configuring Visual Studio 2010-2017 help manually

This chapter gives the information how to install LightningChart Ultimate help content manually. This information is needed if Visual Studio 2010-2017 does not have any local help content installed. When installing LightningChart Ultimate and there isn't any local help content installed, LightningChart Ultimate's help will not install.

These steps allow to view LightningChart Ultimate's help from Visual Studio 2010-2017. Either press F1 on LightningChart Ultimate's classes, properties etc. or use Microsoft Help Viewer to browse the help content.

2.4.1 Visual Studio 2010

Follow these steps to manually install LightningChart Ultimate help content on Visual Studio 2010:

1. Open Visual Studio 2010.
2. Select **Help -> Manage Help Settings**.
3. On Help Library Manager, click **Settings** link.
4. Make sure that **I want to use local help** is selected.
5. If **I want to use local help** is selected, click **Cancel** to go back to Help Library Manager. Otherwise click **OK**.
6. Click **Install content from disk** link.
7. Click **Browse** button and go to the folder where LightningChart Ultimate is installed, by default the path is **C:\Program Files (x86)\Arction\LightningChart Ultimate SDK v.8\MSHelpViewer**.
8. Select **HelpContentSetup.msha** and click **Open** button.
9. Click **Next** button.

10. Next to LightningChart Ultimate Help there is **Add** link. Click it and make sure that **Status** column value changes to **Update Pending**.
11. Click **Update** button. If Help Library Manager asks if you want to proceed, click **Yes** button. Help library update begins.
12. After help library is updated, click **Finish** button to close Help Library Manager.

2.4.2 Visual Studio 2012-2017

Follow these steps to manually install LightningChart Ultimate help content on Visual Studio 2012-2017:

1. Open Visual Studio 2012, 2013, 2015 or 2017.
2. Select **HELP -> Add and Remove Help Content**.
3. After Microsoft Help Viewer starts, select **Manage Content**.
4. Select **Disk** under **Installation source**.
5. Click the button with three dots to browse files.
6. Go to the folder where LightningChart Ultimate is installed, by default the path is **C:\Program Files (x86)\Arction\LightningChart Ultimate SDK v.8\MSHelpViewer**
7. Select **HelpContentSetup.msha** and click **Open** button.
8. Next to LightningChart Ultimate Help there is **Add** link. Click it and make sure that **Status** column value changes to **Add pending**.

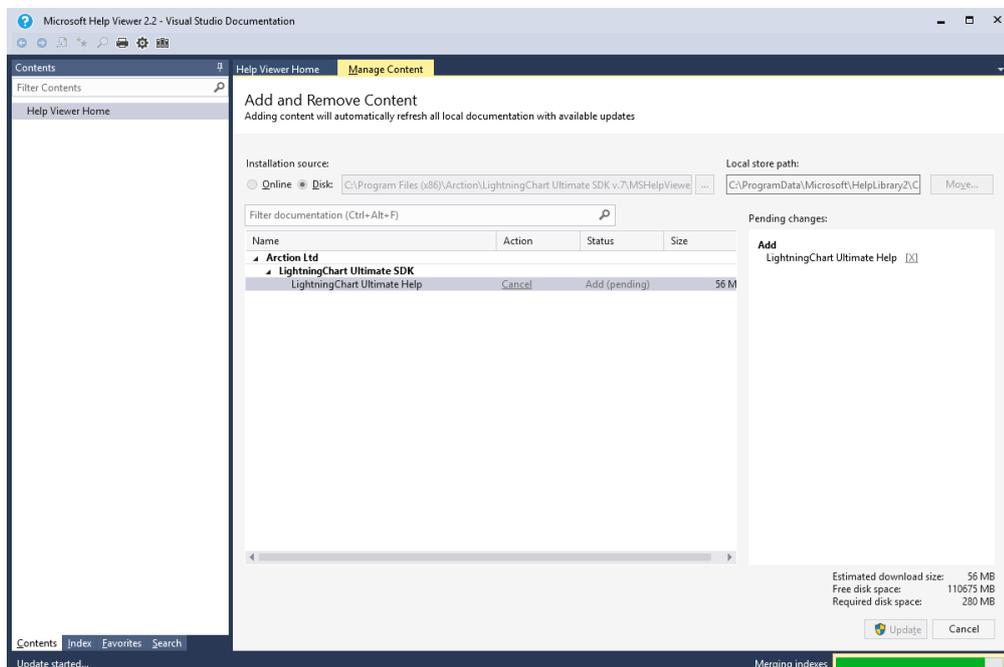


Figure 2-1. Adding LightningChart Ultimate help

9. Click **Update** button. If Help Library Manager asks if you want to continue, click **Yes** button. Help library update begins.
10. After help library is updated, Microsoft Help Viewer can be closed.
11. In Visual Studio Menu / Help, select Set **Help Preference : Launch in Help Viewer**

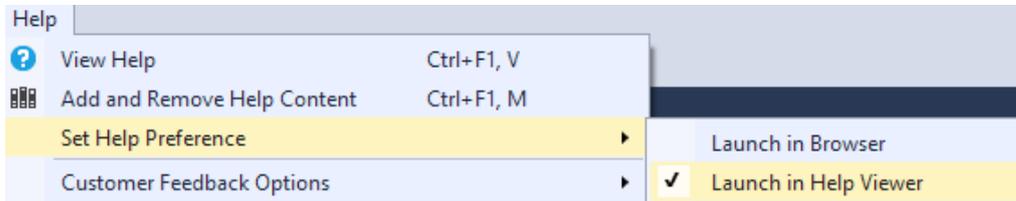


Figure 2-2. Setting help preference.

2.5 Code parameters and tips by Visual Studio IntelliSense

IntelliSense may not show code hints when typing LightningChart related code, if the LightningChartUltimate.dll file is referenced from Global Assembly Cache and the controls are not installed by the automatic toolbox installer. Remove the LightningChartUltimate.dll file from References list of the project. Then add it again by browsing from the install directory (typically **C:\Program files (x86)\Arction\LightningChart Ultimate SDK v.8\LibNet4**)

2.6 Selecting target framework

In C# project, the framework selection can be made in **Project -> Properties -> Application -> Target framework**.

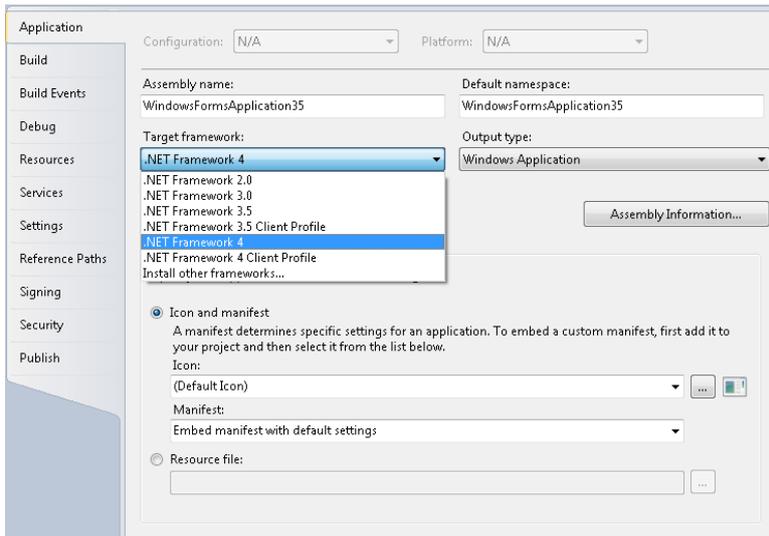


Figure 2-3. Selecting target framework in C# project.

In Visual Basic project, the framework can be selected in **Project -> Options -> Compile -> Advanced compile options -> Target framework**.

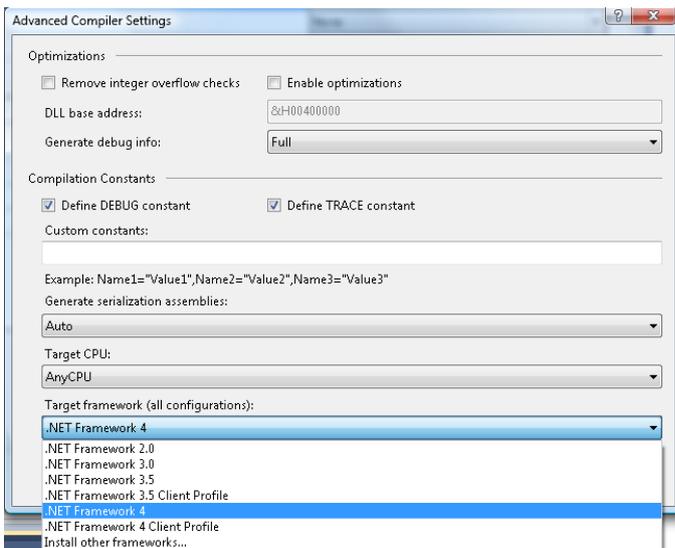


Figure 2-4. Selecting target framework in Visual Basic project.

Select **.NET Framework 4**, **.NET Framework 4 Client Profile** or **.NET Framework 4.5**, or **4.6**).

The LightningChart Ultimate SDK controls will appear in the Visual Studio toolbox only if the correct .NET framework is selected.

3. License management

3.1 Adding license

Manage licenses by running the License Manager application from the Windows start menu: Programs / Arction / LightningChart Ultimate SDK / **License Manager**.

Arction components use a license key protection system. The components can be used only with a valid license. License has information of:

- Enabled features, such as ViewXY, View3D, ViewPie3D, Maps, ViewPolar, ViewSmith, Volume Rendering, Signal Tools
- WPF / WinForms / Both technologies
- To how many users the license can be activated (1 as standard).
- Subscription expiry date (updates and support ending dates)
- Tech support inclusivity
- Per-developer license or Floating license
- Student license

When dragging an Arction component from Toolbox into an application the first time, a license key is asked in a license manager window. Add all license keys at once from the received license file. Click **Install from file...** and browse the **.alf** file.

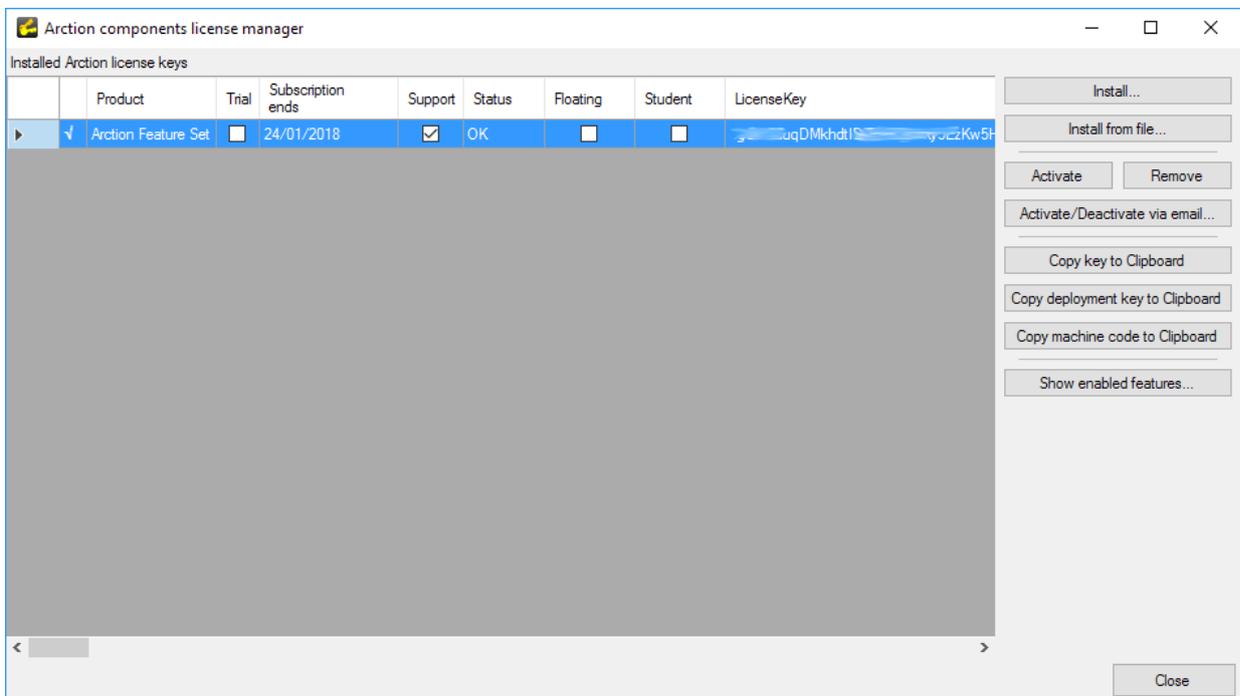


Figure 3-1. Adding license key in LicenseManager.

Per-developer licenses are activated to Arction License Server over internet automatically after adding the license.

If internet connection is not available, use “Activate / Deactivate via e-mail” function.

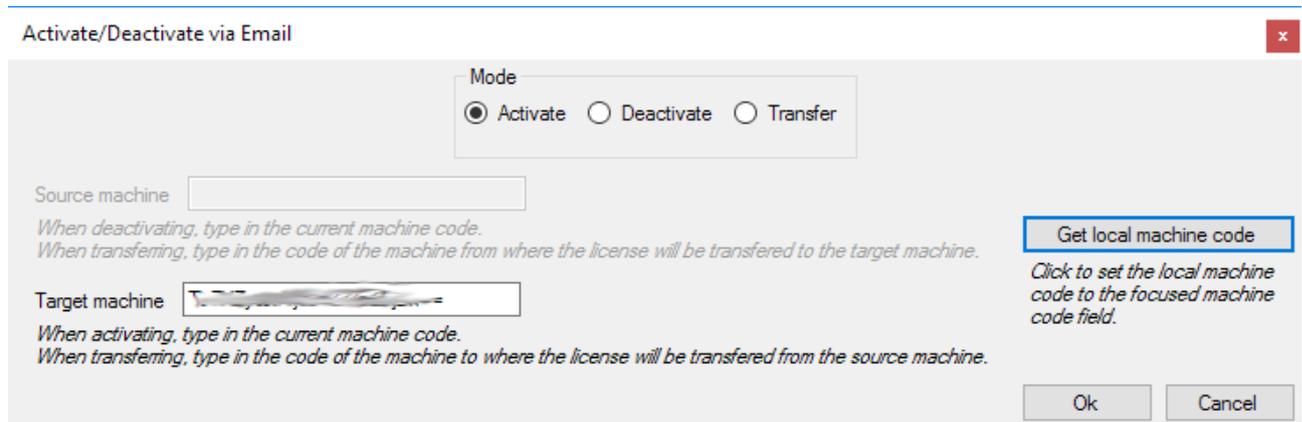


Figure 3-2. Activating a license off-line by e-mail.

Follow on-screen instructions to send an e-mail message to Arction licensing team at licensing@arction.com.

Arction will provide instructions how to install the license off-line. Expect a reply in 2 business days.

Note! Activation/deactivation over telephone is not available, as the key codes contain thousands of characters.

Note! From LightningChart v.7.1 onwards, ChartManager component does not need a license key.

Note! From LightningChart v.8.0 onwards, LIC format license keys are not supported. ALF license is needed. If you haven't received ALF license, please contact Arction.

3.2 Removing a license

License can be removed from the system with Remove button. It needs on-line connection to deactivate it automatically. If internet connection is not available, use “Activate / Deactivate via e-mail” function. Use Mode = Deactivate.

After the license has been deactivated, it can be installed on another computer.

3.3 Extracting Deployment Key

To be able to run LightningChart applications in computers the software is deployed into, a Deployment Key has to be applied in code. Deployment Key can be extracted from a license key by pressing **Copy deployment key to Clipboard** button.

3.4 Applying Deployment Key in an application

In code, use static **SetDeploymentKey** methods for the components. Call the **SetDeploymentKey** methods somewhere before the components need to be used. The best place to call it would be *static constructor* of the class using the chart, or in the application's main class.

For more detailed instruction on deployment, see chapter 27.

WinForms

Here's an example how to apply the key at the static constructor method of the Program class that is created by default for every WinForms application.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    static class Program
    {
        static Program()
        {
            //Set Deployment Key for Arction components
            string deploymentKey = "VMaLgCAA06k01RgiNIBJABVcG.R..Kikfd...";

            Arction.WinForms.Charting.LightningChartUltimate.SetDeploymentKey(deploymentKey);
            Arction.WinForms.SignalProcessing.SignalGenerator.SetDeploymentKey(deploymentKey);
            Arction.WinForms.SignalProcessing.AudioInput.SetDeploymentKey(deploymentKey);
            Arction.WinForms.SignalProcessing.AudioOutput.SetDeploymentKey(deploymentKey);
            Arction.WinForms.SignalProcessing.SpectrumCalculator.SetDeploymentKey(deploymentKey);
            Arction.WinForms.SignalProcessing.SignalReader.SetDeploymentKey(deploymentKey);
        }

        // Rest of the class ...
    }
}
```

WPF

Here's an example how to apply the key in the beginning of App.xaml.cs, at the static constructor of the *App* class.

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Windows;
using Arction.Wpf.SignalProcessing;

namespace WpfApplication1
{
    /// <summary>
    /// Interaction logic for App.xaml
    /// </summary>
    public partial class App : Application
    {
        static App()
        {
            // Set Deployment Key for Arction components
            string deploymentKey = "- DEPLOYMENT KEY FROM LICENSE MANAGER
            GOES HERE-";

            // Setting Deployment Key for fully bindable chart
            Arction.Wpf.BindableCharting.LightningChartUltimate
            .SetDeploymentKey(deploymentKey);

            // Setting Deployment Key for semi-bindable chart
            Arction.Wpf.SemibindableCharting.LightningChartUltimate
            .SetDeploymentKey(deploymentKey);

            // Setting Deployment Key for non-bindable chart
            Arction.Wpf.Charting.LightningChartUltimate
            .SetDeploymentKey(deploymentKey);

            // Setting of deployment key to other Arction components
            SignalGenerator.SetDeploymentKey(deploymentKey);
            AudioInput.SetDeploymentKey(deploymentKey);
            AudioOutput.SetDeploymentKey(deploymentKey);
            SpectrumCalculator.SetDeploymentKey(deploymentKey);
            SignalReader.SetDeploymentKey(deploymentKey);
        }
    }
}
```

Note! Without setting Deployment Key in the application, it enters into 30 days trial mode in the target machine (applies to computers where a Development license key hasn't been installed).

3.5 Running with Deployment Key on development computer

When running an application, in which a deployment key has been applied with **SetDeploymentKey**, on a computer where a development license has been installed to, the library **prioritizes the development license key**. It might lead into user or debugging confusion when deployment key has higher level of features included (e.g. Gold pack) than locally installed license (e.g. Silver pack). Developer must be aware of this limitation.

Arction recommends all licenses to be of same type within the whole team.

3.6 Running with debugger

With Deployment Key set correctly, when running the project from Visual Studio with debugger attached, and no development license key is found from the system, the chart enters slow rendering mode, max FPS is ~1, and the chart shows message text over the chart.

Direct developing and debugging with LightningChart without developer license key, is forbidden by LightningChart EULA.

3.7 Trial period

The trial period is usable for 30 days. After that, a license must be purchased to continue using the product. All projects built with a trial license will work also after updating to proper license. A trial version nag message will be shown when running the chart application built with a trial license.

3.8 Floating licenses

Floating licenses can be installed to unlimited count of computers. Number of concurrent developers has been configured by Arction. Only the purchased count of concurrent users can develop with LightningChart at same time. After a developer finishes LightningChart development, there's about 10-15 minutes timeout until another developer can start using it.

Deployment key must be set similarly than with per-developer licenses.

Floating licenses are controlled by Arction Licensing Server by default. Continuous internet connection is required while developing.

Customer-side floating license controller is also available. Development computers connect to a service running in customer's organization via local area network. On-line communication with Arction or other parties doesn't take place. With licenses, Arction provides separate instructions for installing the controller service and floating licenses.

4. LightningChart Ultimate component

4.1 Adding from toolbox into Windows Forms project

Add **LightningChart Ultimate** control from the toolbox into the form. The chart appears in the form and its properties are shown in **Properties window**.

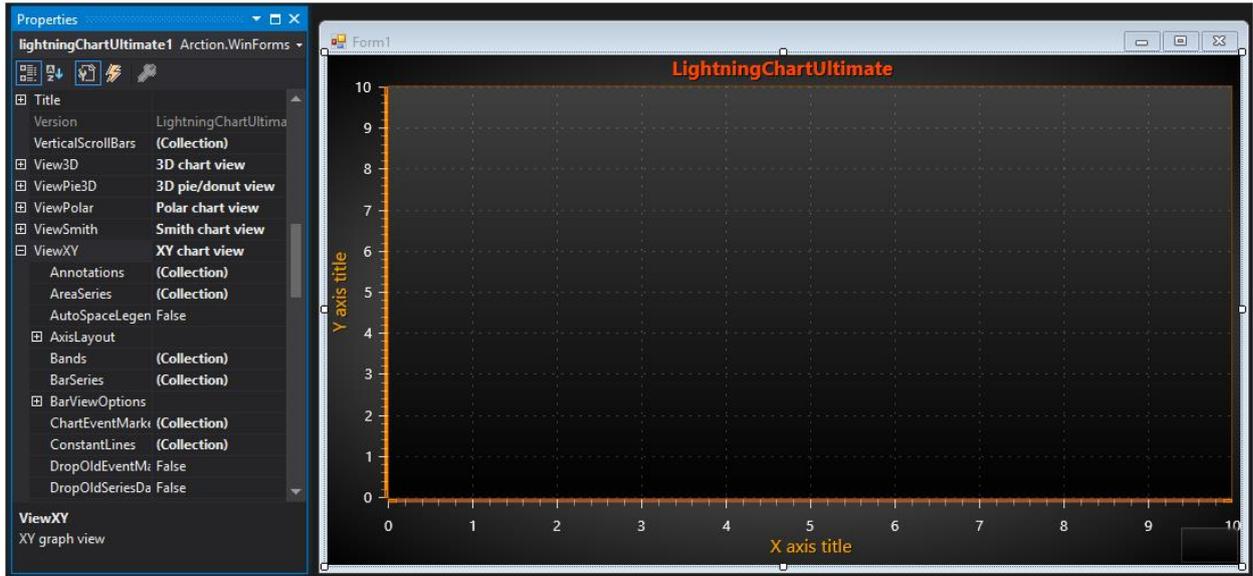


Figure 4-1. LightningChart Ultimate control added into Windows Forms designer.

4.1.1 Properties

The properties can be modified freely. Also, new series and other objects can be inserted in their collections. Series data points must be given by code.

4.1.2 Event handlers

Event handlers of the chart main level can be assigned with the property grid. For objects that have been added to the collections, events handlers must be assigned in code.

4.1.3 Best practices concerning version updates

Chart properties' data is serialized in **.resx** file in the Visual studio project. LightningChart Ultimate API tends to change a little bit with version updates which may lead into incompatible serialization for the new version to exist in the **.resx** file.

For easier updates, it's strongly recommended to create the chart object, add all series, event handlers etc. in code. The project then loads correctly and possible errors are shown in the compile time making it easy to fix them compared to fixing **.resx** file. With **.resx** file, some property definitions may be lost, but in code, they are always specified.

4.2 Adding from toolbox into WPF project

Add **LightningChart Ultimate** Semi-bindable or Bindable WPF control from the toolbox to Window or another container. The chart appears in the designer and its properties are shown in **Properties** window. XAML editor shows the contents and modifications to the chart default properties.

4.2.1 Properties

The properties can be modified freely, and new series and other objects can be inserted into their collections.

- In Semi-bindable chart, series data points must be given by code
- In Bindable chart, series data points can be given by binding data to the series, or in code behind

4.2.2 Event handlers

Event handlers of the chart main level can be assigned with the property grid. For objects that have been added to the collections, events handlers must be assigned in code.

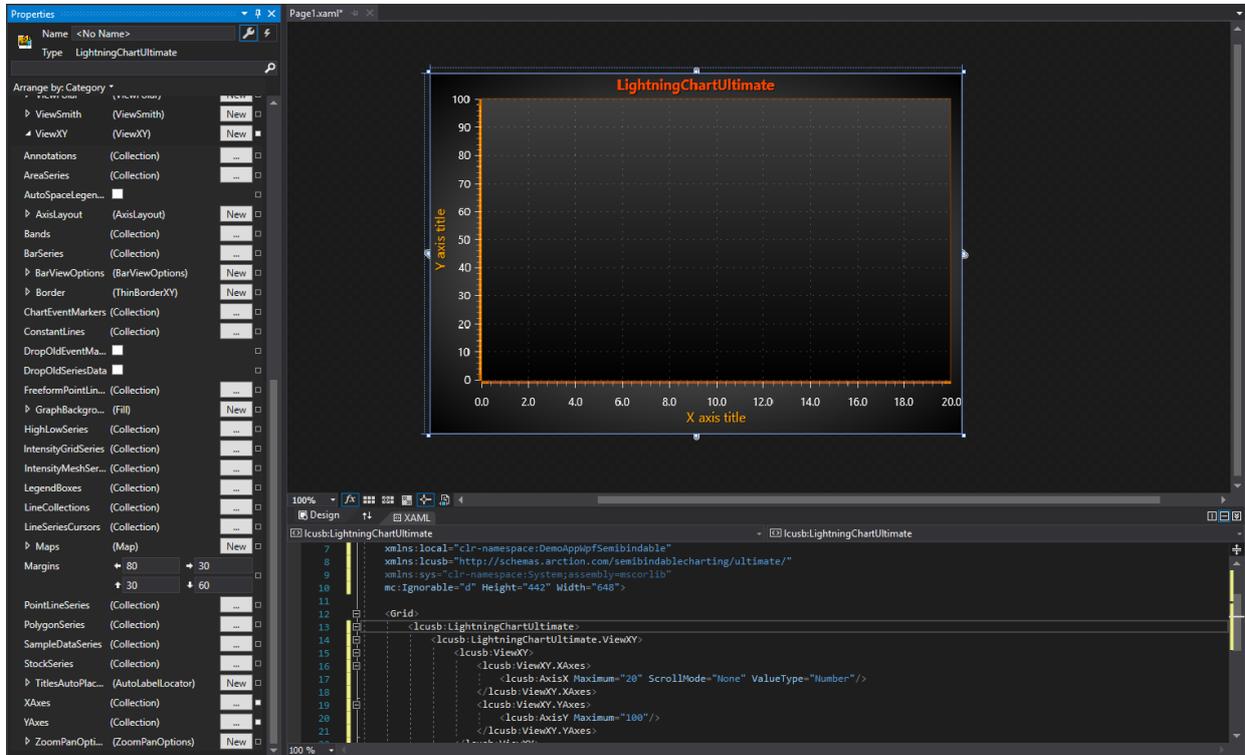


Figure 4-2. LightningChart Ultimate control added into WPF designer.

4.3 Adding into Blend WPF project

In **Projects** tab, go to **References**. Right-click and select **Add reference...** Browse Arction.WPF.LightningChartUltimate.dll from *c:\program files (x86)\Arction\LightningChart Ultimate SDK v.8\LibNet4*.

Go to **Assets** tab. Write "Lightning" in the Search box. **LightningChart Ultimate** row can be found in the search results. Drag-drop the object into the WPF window.

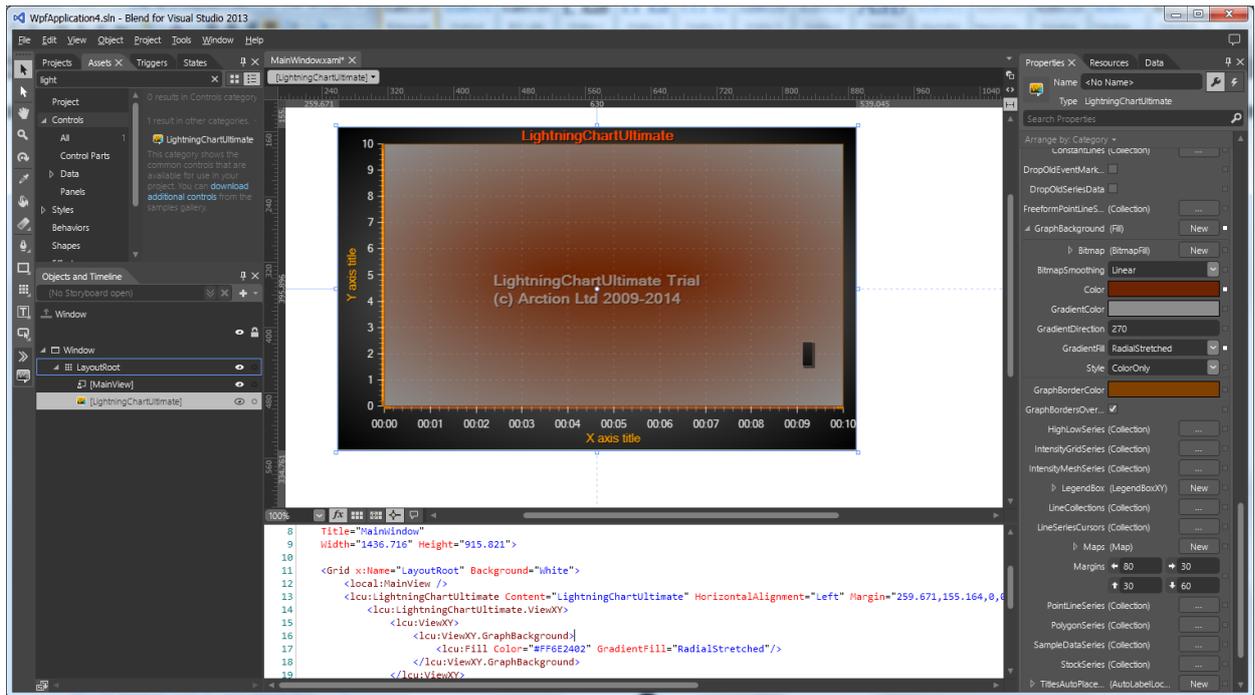


Figure 4-3. LightningChart Ultimate control added into Blend For Visual Studio 2013 designer.

4.3.1 Best practices concerning version updates

Chart properties data is stored in XAML. New versions may have slightly different property set, which can cause the LightningChart Ultimate object not to appear in the designer. Relevant XAML modifications are then needed. The XAML tags tree may be huge and editing it may be quite difficult.

For easier updates, it's strongly recommended to create the chart object and set its layout and alignment relevant properties in designer. Set everything else in code. Alternatively, create the chart object in code as well.

4.3.2 Preventing blurring of the chart

This is a common feature of WPF and not related to the chart itself but becomes clearly visible in accurate rendering of LightningChart.

To prevent the chart to appear blurred, set **UseLayoutRounding = True** of the control that is **parent** to the chart. The chart may still appear blurred in the designer but will look sharp when running the application. The parent control can be for example **Grid, Canvas, DockManager** etc.

4.4 Object model

The best way to learn the object model of LightningChart is by using *Properties editor* of Visual Studio.

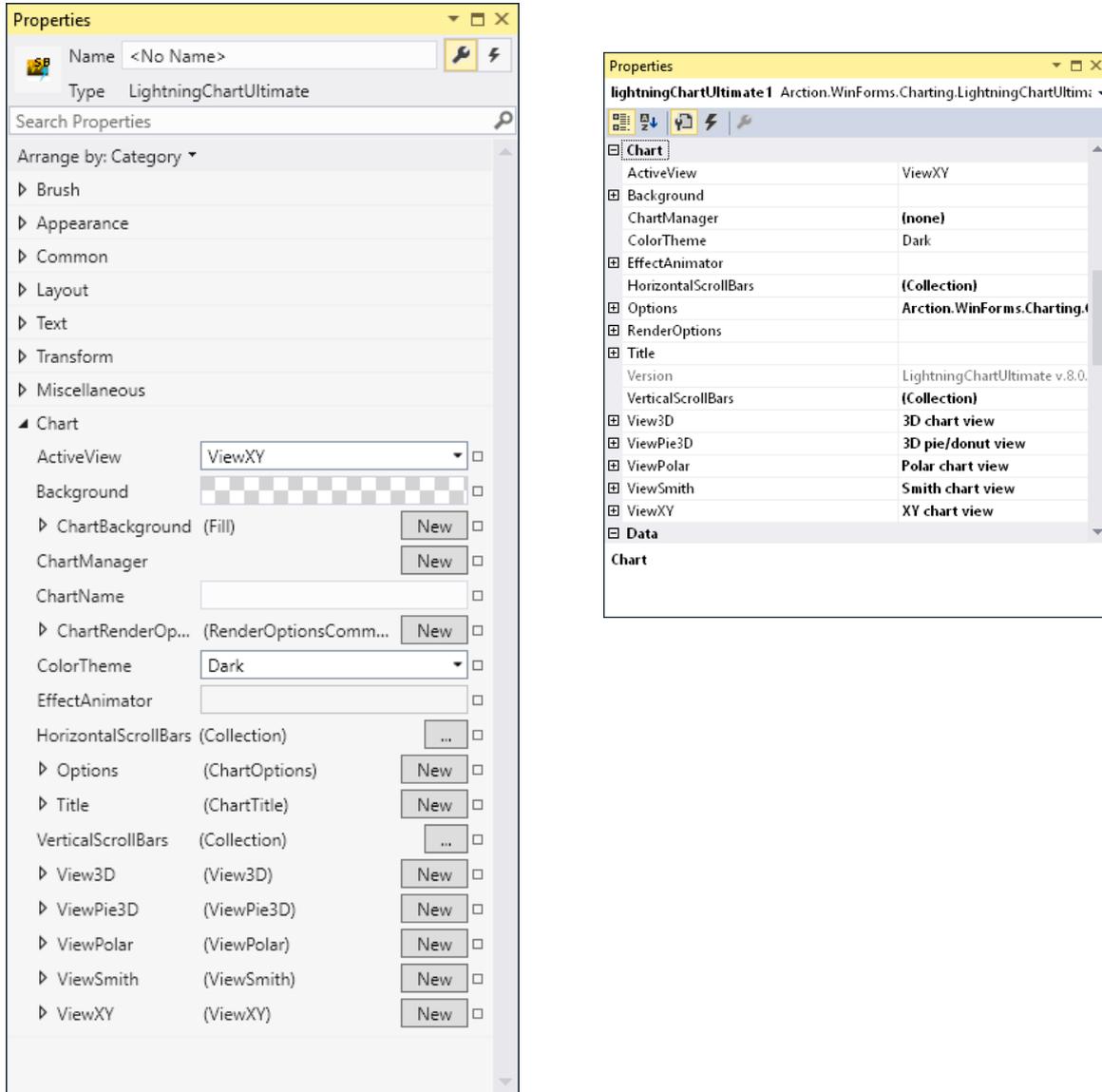


Figure 4-4. LightningChart specific properties can be found under *Chart* category in both Windows Forms and WPF Properties window. By expanding the nodes, or in WPF creating new objects, a huge set of properties can be seen.

4.4.1 Differences between Windows forms and WPF

The property tree and object model between Windows Forms and WPF are almost identical, regarding the Chart category. The main differences are:

	Windows Forms	WPF
Rendering options property	RenderOptions	ChartRenderOptions
Background fill property	Background	ChartBackground
Fonts	System.Drawing.Font	Arction.WPF.LightningChartUltimate.WPFFont
Colors	System.Drawing.Color	System.Windows.Media.Color

Table 4-1. Differences between Windows forms and WPF

In the following chapters, Windows forms property names are referred unless otherwise denoted.

4.5 LightningChart Views

LightningChart has these main views:

- ViewXY (see chapter 5)
- View3D (see chapter 6)
- ViewPie3D (see chapter 8)
- ViewPolar (see chapter 9)
- ViewSmith (see chapter 10)

The visible view can be changed by setting **ActiveView** property. The default view is ViewXY.

4.6 View and zooming area definitions

LightningChart views contain several various areas determined by the information they hold. The areas can be seen as two-dimensional rectangles based on the content of the view. These definitions are uniform regardless of the view type. They are used especially in zooming operations to determine which areas of the chart will be shown.

-ChartArea/ViewArea: The whole area including the chart and the margins.

-MarginRectangle: MarginRectangle (or MarginRect) includes the area inside the margins.

-GraphArea: The area defined by the axis ranges. Contains major and minor grids. The data is drawn in this area, unless some data values exceed the axis ranges.

-Background area / circle: Is mostly the same as the GraphArea. Contains also the parts of the graph outside the axis ranges and the grids.

-LabelsArea: The area consisting of the graph and the axis labels. Ignores the data.

-Data: The area containing only the data. Defined by the minimum and the maximum values of the data.

-DataAndLabelsArea: Data and LabelsArea combined. All data, axes, labels and markers are included.

-Border: A customizable, one-pixel wide rectangle, which indicates the location of the margins. Its visibility can be changed by disabling/enabling it.

-Margins: Margins are empty spaces around the graph area. Most of the contents of the view are fitted inside the margins and clipped outside them.

-ZoomPadding: The space left between the margins and another, pre-defined area after a zooming operation (see chapter 6.18.3). It has no effect in ViewXY.

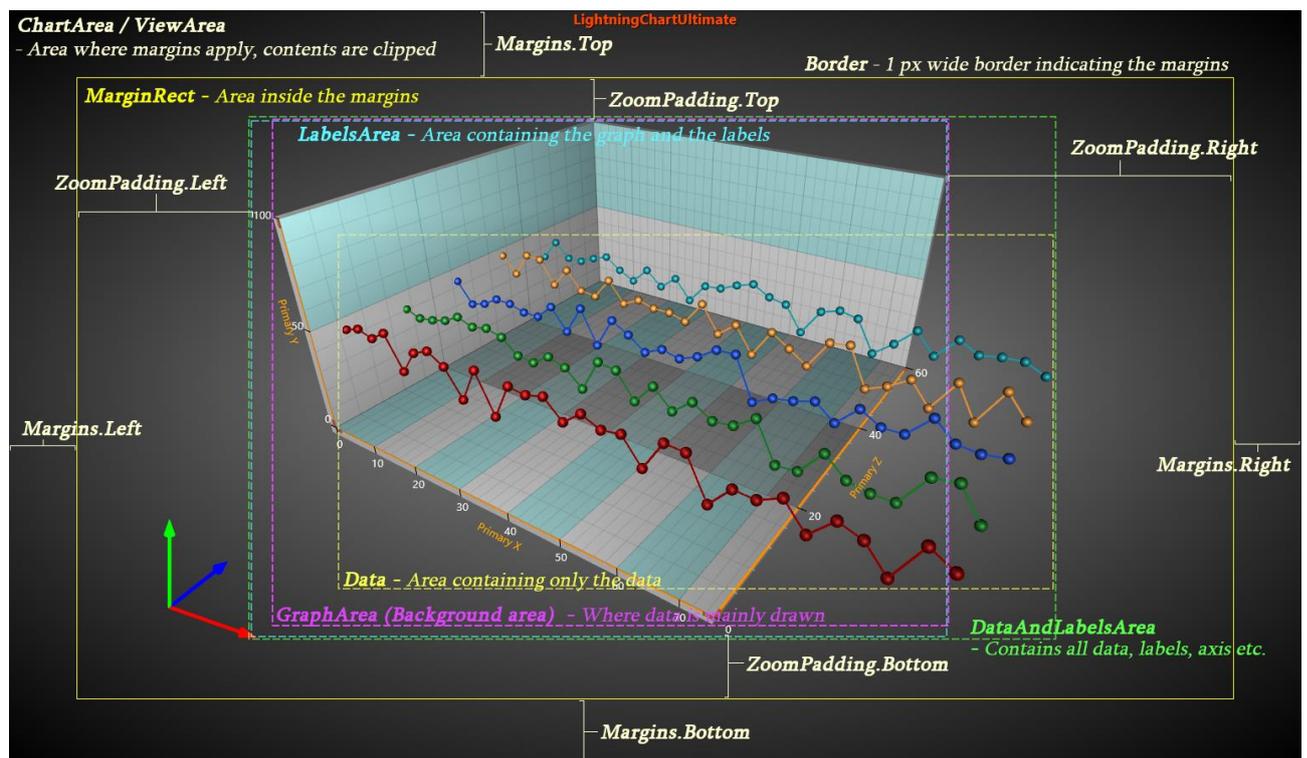


Figure 4-5. View and zooming area definitions

4.7 Setting background fill

All views have a common background fill.

- Use ***chart.Background*** in WinForms.
- Use ***chart.ChartBackground*** in WPF.

The background fill supports:

- Solid color fills. Set ***GradientFill = Solid*** and use ***Color*** to define the color.
- Gradient fills, going from ***Color*** to ***GradientColor***. Set ***GradientFill = Linear / Radial / RadialStretched / Cylindrical***. Use ***GradientDirection*** to control the fill direction in Linear and Cylindrical gradients.
- Bitmap fills, with different tiling and stretching options. Bitmap tint and alpha also are supported to, to make translucent bitmap fills.



Figure 4-6. Setting Background background, underneath ViewXY. GradientFill = Solid, and Color = DimGray.

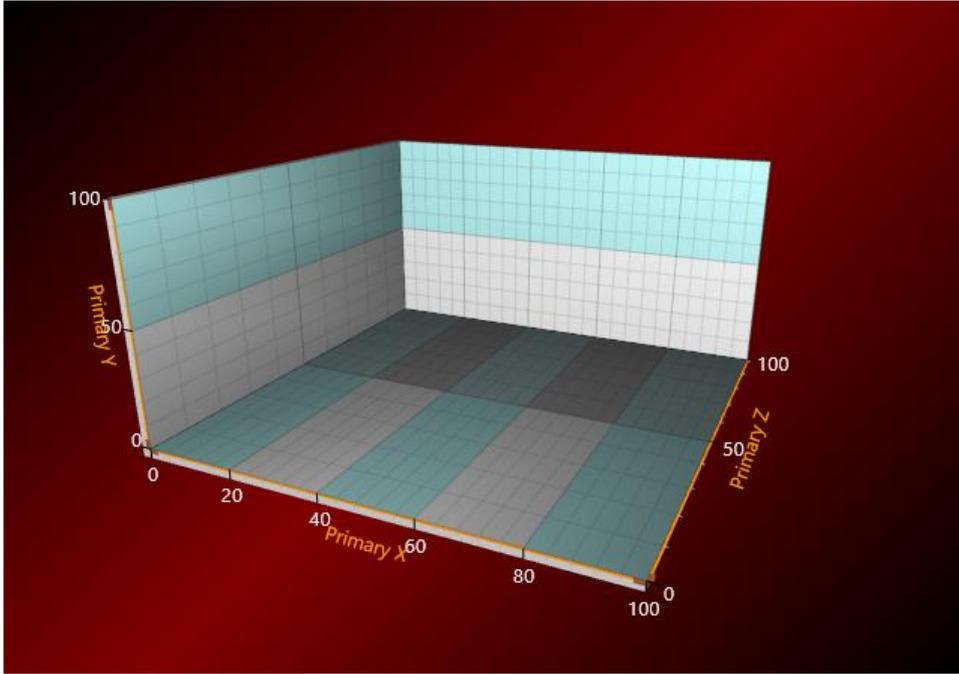


Figure 4-7. Setting Background to gradient cylindrical, under View3D.GradientFill = Cylindrical, Color = Maroon, GradientColor = Black. GradientDirection = -45 degrees.

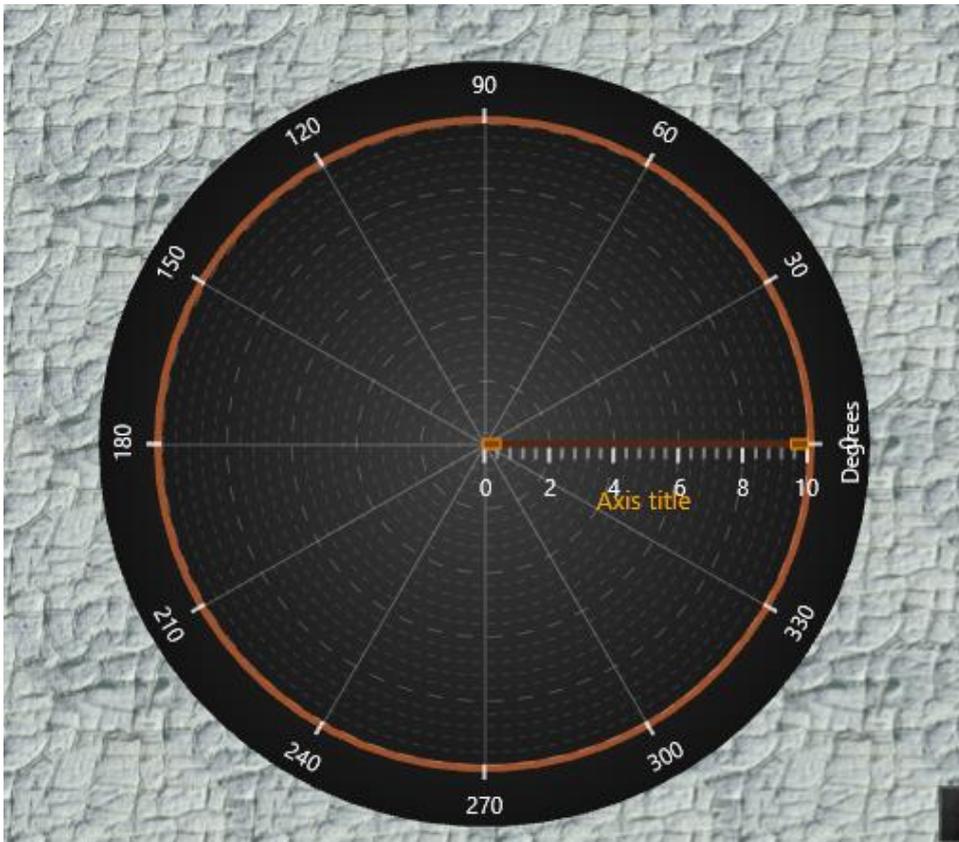


Figure 4-8. Setting Background to tiled bitmap fill. Style = Bitmap, a picture set to Bitmap.Image, and Bitmap.Layout = Tile, under ViewPolar.

4.7.1 Setting transparent background

In WPF, the chart can be made appear transparent, so the objects placed underneath the chart will show through.

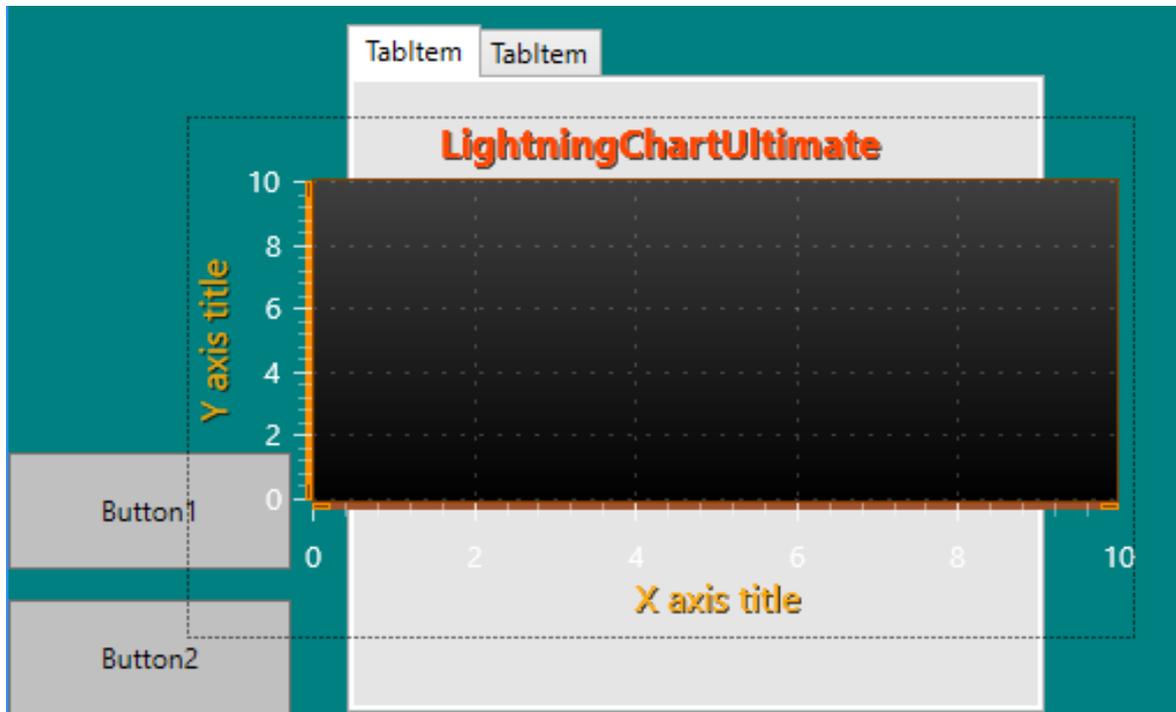


Figure 4-9. Transparent background in WPF chart.

Set `ChartBackground.Color = #00000000` (Black transparent).

Note! Do NOT set 'Transparent' (`#00FFFFFF`). It won't show through.

WinForms does not support transparent background of controls.

4.8 Configuring appearance / performance settings

RenderOptions (**ChartRenderOptions** in WPF) contain properties for configuring appearance and performance.

RenderOptions	
AntiAliasLevel	4
D2DEnabled	True
DeviceType	Auto
FontsQuality	Mid
ForceDeviceCreateOnResize	False
FrameRateLimit	40
GPUPreference	PreferHighPerformanceGraphics
HeadlessMode	False
InvokeRenderingInUIThread	False
LineAAType2D	ALAA
LineAAType3D	QLAA
LineOffset	
RemoteDesktopVendorId	0
UpdateOnResize	True
UpdateOnResizeTimeInterval	1000
UpdateType	Sync
ViewXY	
GDILineSeriesCompression	True
LineSeriesEnhancedAntiAliasing	Off
WaitForVSync	False

Figure 4-10. RenderOptions properties.

DeviceType

Auto is an alias to AutoPreferD11 option. This is the default setting.

AutoPreferD9 prefers DirectX9 hardware rendering, and automatically selects device in this order: HW9 -> HW11 -> SW11 -> SW9 based on availability. Falls back to WARP (SW11) software rendering when hardware is not available.

AutoPreferD11 prefers DirectX11 hardware rendering, and automatically selects device in this order: HW11 -> HW9 -> SW11 -> SW9 based on availability. Falls back to WARP (SW11) software rendering when hardware is not available. **Use this as a general high-performance and best appearance setting.** Visual appearance is better than with DirectX9 renderer.

HardwareOnlyD9 uses hardware 9 rendering only.

HardwareOnlyD11 uses hardware 11 rendering only.

SoftwareOnlyD11 uses DirectX11 WARP, very fast when compared to DirectX9 reference rasterizer, but slower than hardware options)

SoftwareOnlyD9 uses DirectX9 reference rasterizer (very slow)

None if chart is hidden, or inactive in background, setting **DeviceType** to **None** will free graphics resources to other charts.

GPUPreference

A selection applicable to machines with dual graphics adapter systems, mainly laptops having integrated low-performance Graphics Processing Unit (GPU) in the CPU/chipset, and higher performance graphics GPU (e.g. AMD or Nvidia).

SystemSetting uses options selected in the graphics settings of Windows, or AMD or Nvidia control panel.

PreferHighPerformanceGraphics uses high-performance GPU if it exists in the system. Gives better performance in general but may lead into higher energy consumption.

PreferLowPowerGraphics uses slower integrated GPU, even if high-performance GPU has been installed on the system.

By default, **PreferHighPerformanceGraphics** is the preferred option. Keep it selected to get the best performance.

FontsQuality

Low gives best performance, the fonts are not anti-aliased. Select font typeface carefully to get acceptable appearance.

Mid gives almost similar performance than **Low**. Has simple anti-aliasing around the fonts. This is the default setting.

High gives best appearance but has a significant performance hit.

Note: Transparent background is not applicable for DirectX 11 rendering with **High** quality setting. For DirectX9 it works. This is a rendering technology limitation.

AntiAliasLevel

The overall scene anti-aliasing factor. Availability depends on the hardware. Higher values give better appearance, but with reduced performance. Set 0 or 1 to maximize the performance.

WaitForVSync

Recommendation: keep as default value. When enabled, holds rendering until display's next refresh is taking place (e.g. next multiple of 1/60 s). Only recommended temporarily e.g. when synchronization with external screen capture application is used to prevent striping, or when image on the screen in top of the screen is not in sync with bottom of the screen. It may show as broken waveform data. Significant performance hit when enabled, especially in WPF.

UpdateType

Sync (default): Chart is updated synchronously. Chart gets updated either after the last ***EndUpdate()*** call, or when setting a property (or calling a method) causes some changes in the Chart. Property change (without ***BeginUpdate...EndUpdate()***) leads to immediate new frame rendering.

Async: Chart is updated on async fashion. The chart will update as fast as possible after property changes, but chart will render a new frame at some later point. This might make it easier to use chart on some cases.

LimitedFrameRate: Frame rate is limited to value specified in `FrameRateLimit` property. 0 = unlimited. E.g. if max. 10 refreshes / second is wanted, set 10. This is similar to the Async option but prevents new frames to be rendered right after first one, thus reducing framerate, but sparing system resources.

Note! Ensure correct thread handling also in LimitedFrameRate and Async modes. If chart updates asynchronously, and chart properties are updated at the same time, a conflict may occur and crash the chart or application.

InvokeRenderingInUIThread

When using a background thread in the application, all UI updates from the thread must go through `Invoke` (***Control.Invoke()*** in WinForms, and ***Dispatcher.Invoke()*** in WPF).

The rendering part will use internal `Invoke` to UI thread, when enabled.

The default value is ***False***, as setting properties and calling methods in a thread-safe way should also be take care of, even when this property is enabled, to prevent thread collision in internal states of the chart.

HeadlessMode

Setting this to ***True*** allows using the chart in a background service, console application or other application without user interface. See chapter 22.

4.9 DPI handling

By default, WPF applications are DPI (Dots Per Inch) aware whereas WinForms apps are not. Also, DPIs are used instead of pixels to measure sizes. LightningChart does not support per-monitor DPI awareness but does system awareness, meaning that WPF apps are DPI system aware. Default DPI in WinForms is 72, but it is worth noting that if wpf .dll files are loaded, the value changes to 96.

However, LightningChart will not automatically resize when moved to another screen with different DPI settings. To enable resizing, ***AllowDPIChangeInduceWindowsResize*** property under ***ChartOptions*** needs to be set ***true***. Alternatively, user can register to ***OnDPIChanged*** event and change its ***allowWindowResize*** attribute. These have no effect in WinForms.

4.9.1 DpiHelper class

LightningChart has ***DpiHelper*** class, which contains helpers on DPI related issues.

DpiAware states if the system process is DPI aware or not. However, it is currently not possible to distinguish between system aware and per-monitor aware.

DpiXFactor/ DpiYFactor is an effective Zoom factor of the system DPI of the screen width/height. Factor that describes how many real pixels there are per one DPI in X/Y direction.

DipToPx and ***PxToDip*** methods convert DIPs to pixels and vice versa using system DPI settings. They can convert single points or pixels, or alternatively the size and the position values of a rectangle.

5. ViewXY

ViewXY allows presenting various point-line series, area series, high-low series, intensity series, heat maps, bar series, bands, line series cursors etc. in Cartesian, XY graph format. Series are bound to X and Y axes, and they are using the value range of the assigned axes.

ViewXY can also show geographical maps, see chapter 5.25.

ViewXY	XY chart view
Annotations	(Collection)
AreaSeries	(Collection)
AutoSpaceLegendBoxes	False
> AxisLayout	
Bands	(Collection)
BarSeries	(Collection)
> BarViewOptions	
> Border	Border
ChartEventMarkers	(Collection)
ConstantLines	(Collection)
DropOldEventMarkers	False
DropOldSeriesData	False
FreeformPointLineSeries	(Collection)
> GraphBackground	
HighLowSeries	(Collection)
IntensityGridSeries	(Collection)
IntensityMeshSeries	(Collection)
LegendBoxes	(Collection)
LineCollections	(Collection)
LineSeriesCursors	(Collection)
> Maps	
> Margins	61, 28, 12, 58
PointLineSeries	(Collection)
PolygonSeries	(Collection)
SampleDataSeries	(Collection)
StockSeries	(Collection)
> TitlesAutoPlacement	
XAxes	(Collection)
YAxes	(Collection)
> ZoomPanOptions	

Figure 5-1. ViewXY object tree.

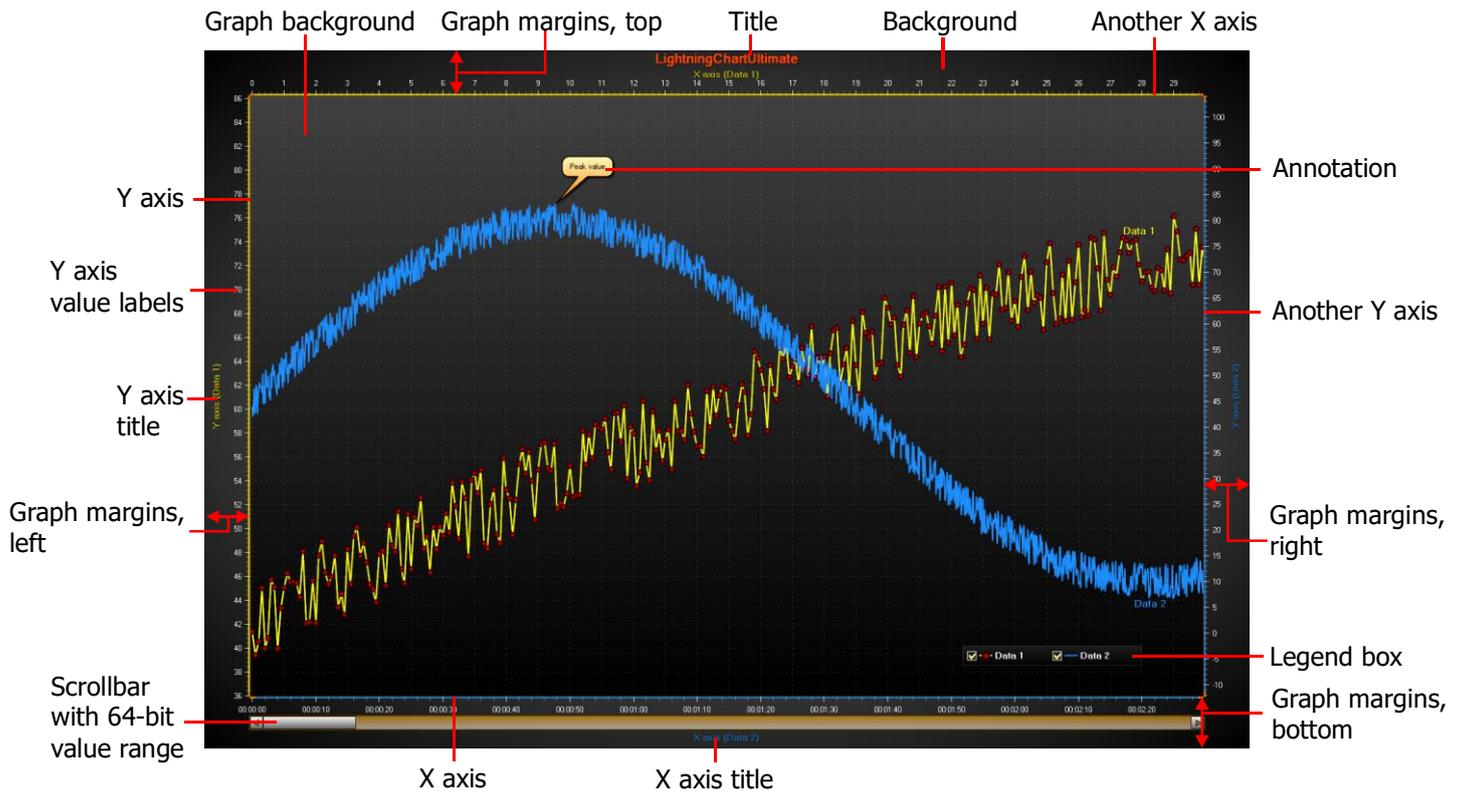


Figure 5-2. A quick overview of ViewXY

Graph margins

Margins are adjusted automatically by default by axis count and their settings. By setting **ViewXY.AxisLayout.AutoAdjustMargins = False**, **Margins** property applies, which allows the margin sizes to be set manually. Set all margins to 0 to make the graph fill the whole view area.

Graph border

A border is drawn around the graph area, in the location of margins. **Border** property can be used to change its color and visibility as well as to determine if it should be rendered behind the series.

Background

Set the background fill with **Background (ChartBackground in WPF)** property. There are plenty of filling options available. See 4.7.

Graph background

Set the graph background fill with **GraphBackground** property. Graph is the area where all grids, series, series cursors, event markers etc. are rendered.

Title

This is the main title for the chart. Set the text, shadow, color, text border, rotation, font, alignment etc. with **Title.Text**, **Title.Shadow...** properties.

Y axes

The vertical axes representing Y values. See chapter 5.2.

X axis

The horizontal axes representing X values. See chapter 5.3.

Annotations

Annotations allows displaying mouse-interactive text labels or graphics anywhere in the chart area. See chapter 5.20.

Legend box

Lists all the series of the chart. See chapter 5.21.

Scrollbar

A scrollbar having unsigned 64-bit value range, to support massive count of sample indices directly. In fact, **HorizontalScrollBars** and **VerticalScrollBars** are collection properties in the chart root level, but they are aware of ViewXY's margins. See chapter 12.

5.1 Axis layout options

The general properties adjusting axis placement, automatic margins etc. can be found in **ViewXY.AxisLayout** properties and sub-properties.

AxisLayout	
AutoAdjustAxisGap	5
AutoAdjustMargins	True
AutoShrinkSegmentsGap	True
AxisGridStrips	None
GridVisibilityOrder	BehindSeries
Segments	(Collection)
SegmentsGap	20
XAxisAutoPlacement	AllBottom
XAxisTitleAutoPlacement	True
XGridStripAxisIndex	0
YAxesLayout	Layered
YAxisAutoPlacement	AllLeft
YAxisTitleAutoPlacement	True
YGridStripAxisIndexLayered	0

Figure 5-3. AxisLayout property tree.

5.1.1 Setting how axes are placed

5.1.1.1 X axis automatic placement

XAxisAutoPlacement controls how the X axes are placed vertically.



Figure 5-4. **XAxisAutoPlacement = AllBottom**. Three X axes added, all are positioned below the graph.

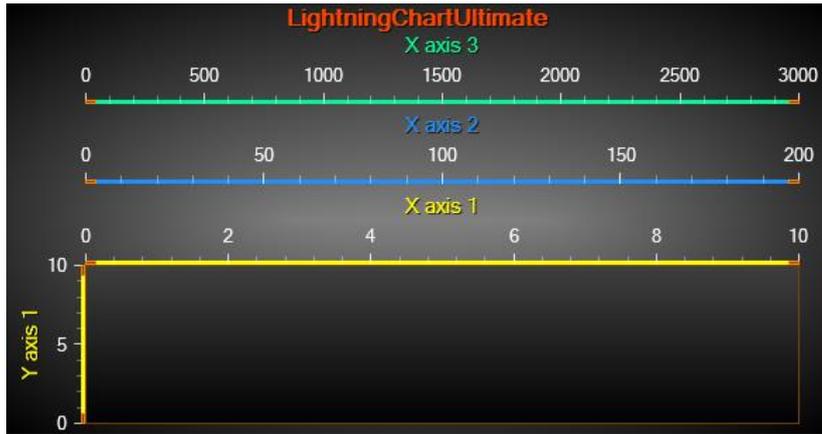


Figure 5-5. XAxisAutoPlacement = AllTop. All X axes are positioned above the graph.

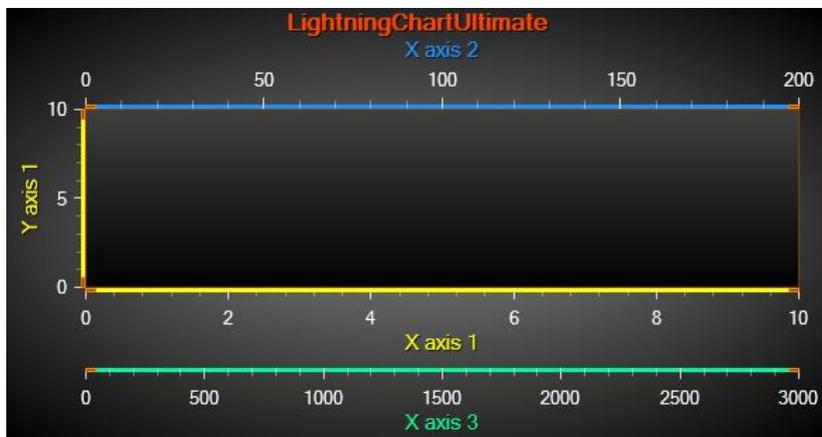


Figure 5-6. XAxisAutoPlacement = BottomThenTop. Axes are distributed below and above the graph, every other axis to the opposite side, starting from bottom.

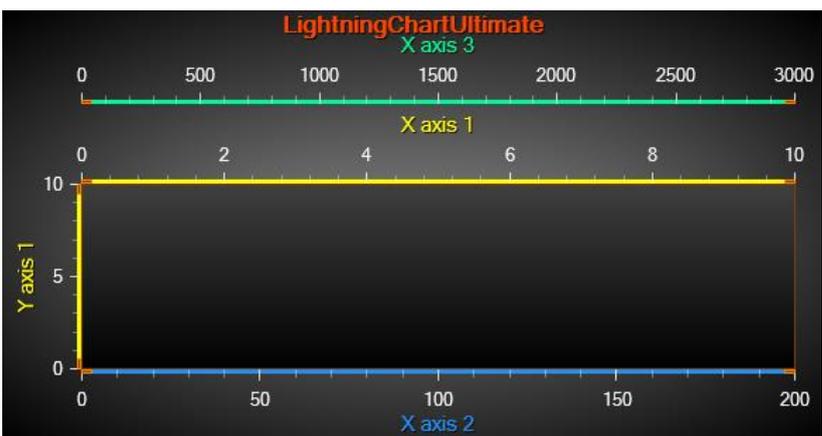


Figure 5-7. XAxisAutoPlacement = TopThenBottom. Axes are distributed below and above the graph, every other axis to the opposite side, starting from top.

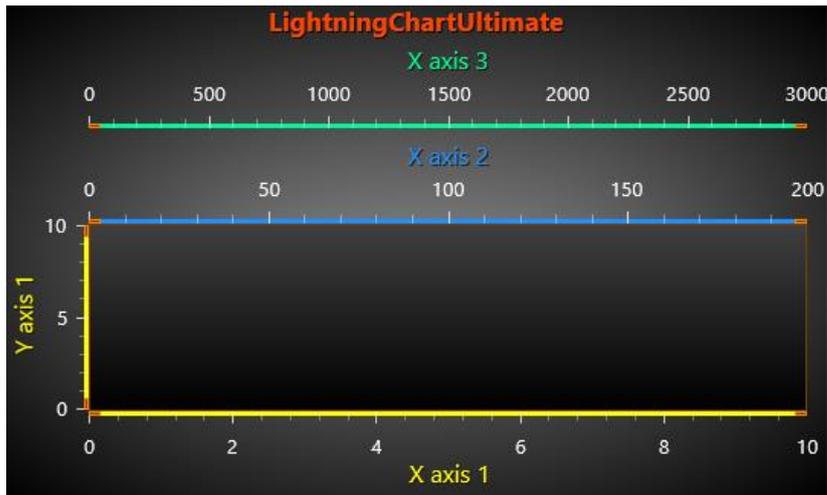


Figure 5-8. XAxisAutoPlacement = Explicit. The axis appears on the side of the selected explicitly. XAxis1 has ExplicitAutoPlacementSide property set to Bottom, whereas XAxis2 and XAxis3 to Top.

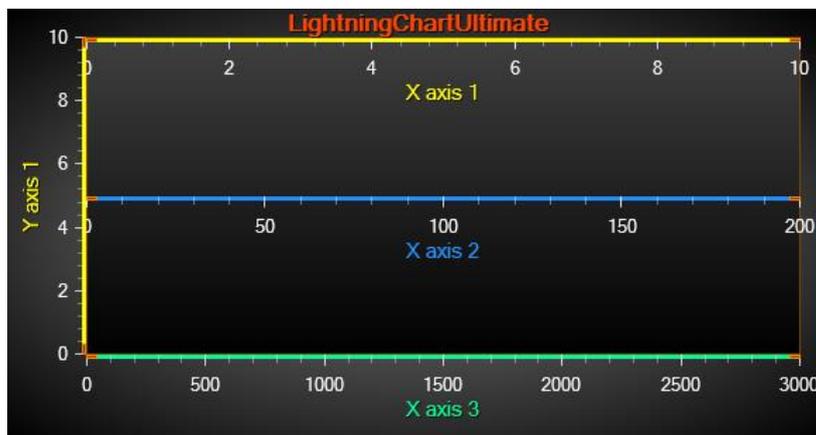


Figure 5-9. XAxisAutoPlacement = Off. Automatic axis placement is disabled, and Position and Alignment properties of each axis apply separately. First axis Position = 0, Second axis Position = 50 and Third axis position = 100.

5.1.1.2 Y axis automatic placement

YAxisAutoPlacement controls how the Y axes are placed horizontally.

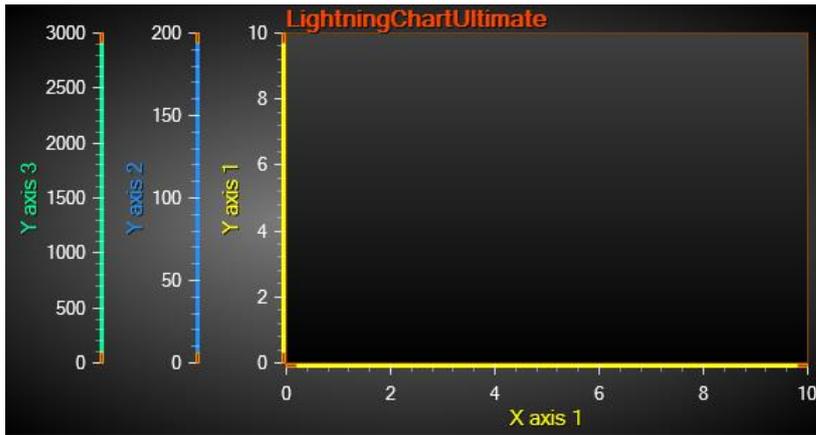


Figure 5-10. YAxisAutoPlacement = AllLeft. Three Y axes added, all are positioned to the left side of the graph.

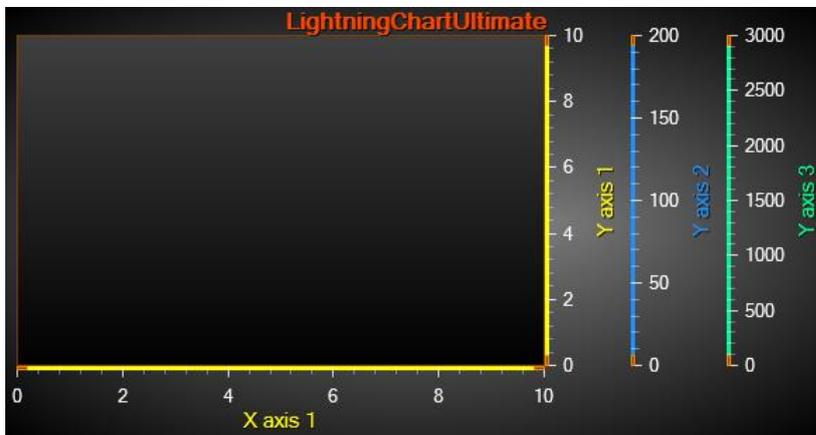


Figure 5-11. YAxisAutoPlacement = AllRight. All Y axes are positioned to the right side of the graph.

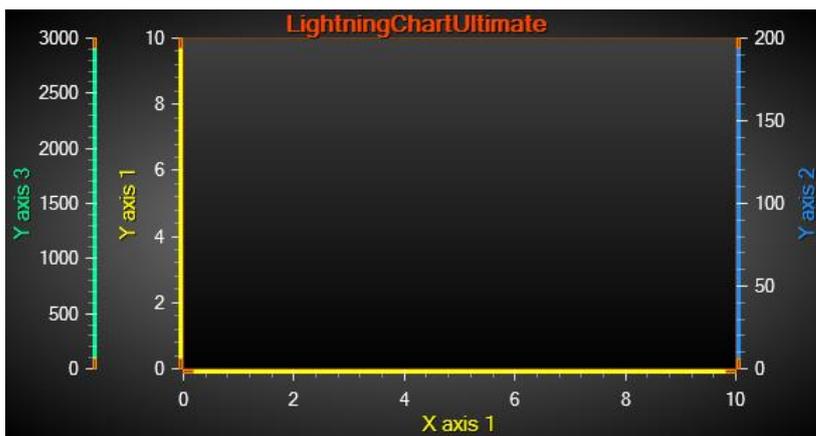


Figure 5-12. YAxisAutoPlacement = LeftThenRight. Axes are distributed to left and right side of the graph, every other axis to the opposite side, starting from the left side.

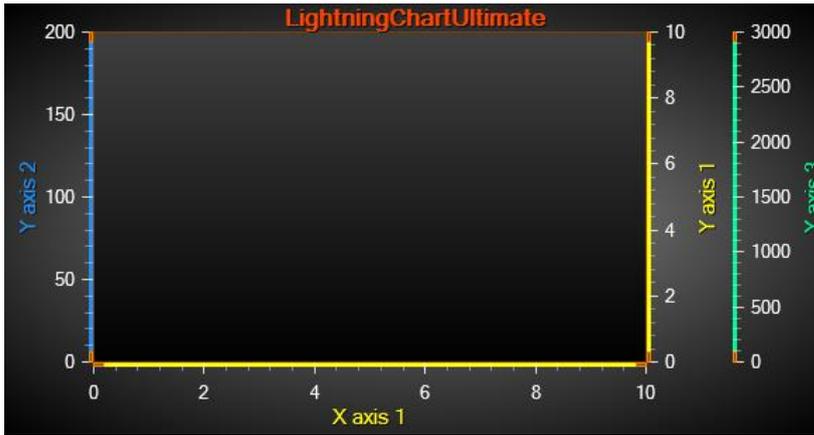


Figure 5-13. YAxisAutoPlacement = RightThenLeft. Axes are distributed to left and right side of the graph, every other axis to the opposite side, starting from the right side.

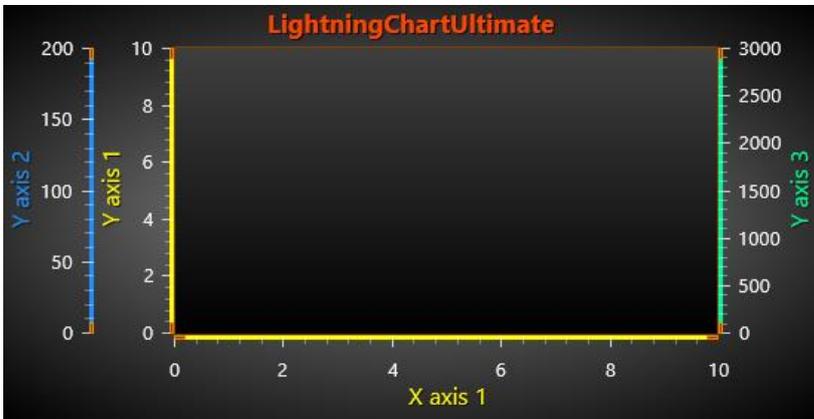


Figure 5-14. YAxisAutoPlacement = Explicit. The axis appears on the side of the selected explicitly. YAxis1 and YAxis2 have ExplicitAutoPlacementSide property set to Left, and YAxis3 to Right.

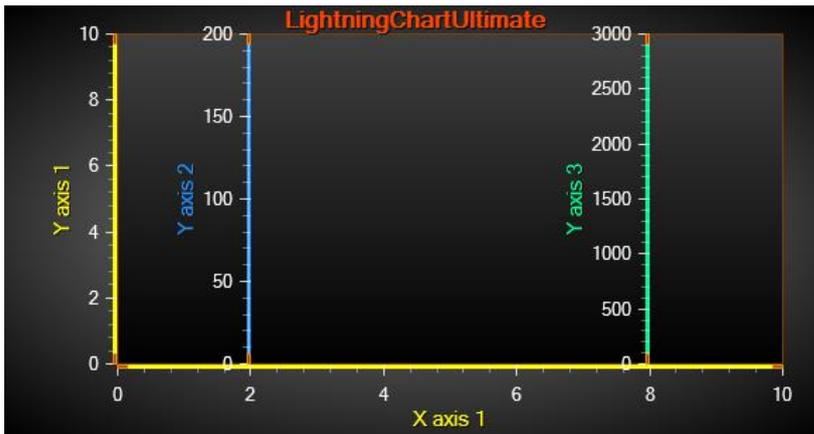


Figure 5-15. YAxisAutoPlacement = Off. Automatic axis placement is disabled, and Position and Alignment properties of each axis apply separately. First axis Position = 0, Second axis Position = 20 and Third axis position = 80.

5.1.2 Graph segments and Y axes placement in them

If there are several Y axes defined, they can be vertically aligned in 3 different ways: **Layered**, **Stacked** and **Segmented**. This can be selected by `ViewXY.AxisLayout.YAxesLayout` property.

5.1.2.1 Layered

In **Layered** view, all the Y axes start from the top of the graph and stretch to the bottom of the graph. The axes and the series bound to them share the same vertical space.

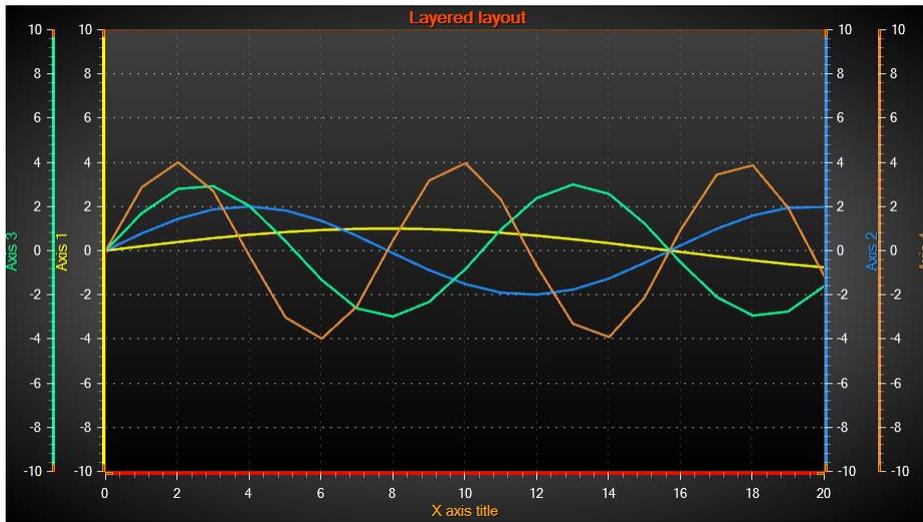


Figure 5-16. An example view of 4 Y axes in `YAxesLayout = Layered`.

5.1.2.2 Stacked

In **Stacked** view each Y axis gets its own vertical space. All Y axes have equal height.

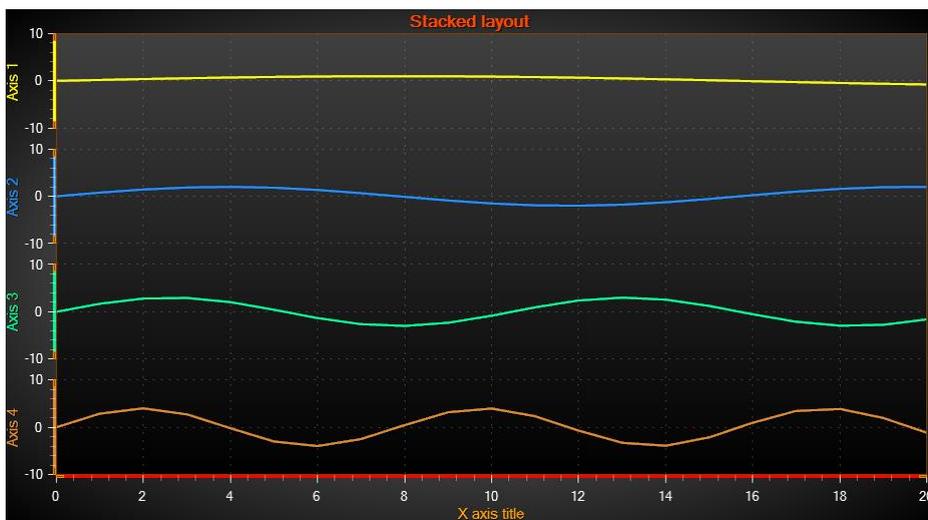


Figure 5-17. An example view of 4 Y axes in `YAxesLayout = Stacked`.

5.1.2.3 Segmented

In Segmented view the vertical space is divided between **Segments**. Each segment can contain several Y axes. The relational height of each segment can be set, and every Y axis within a segment gets the segment's height.

The **Segments** must be created in **AxisLayout.Segments** collection. A segment has only one property, **Height**. It is a relational size versus other segments. It is not defined in screen pixels, as they need to rescale with the chart's size.

The Y axes can be assigned with a segment by setting **yAxis.SegmentIndex** property. The **SegmentIndex** is the index in the **AxisLayout.Segments** collection.

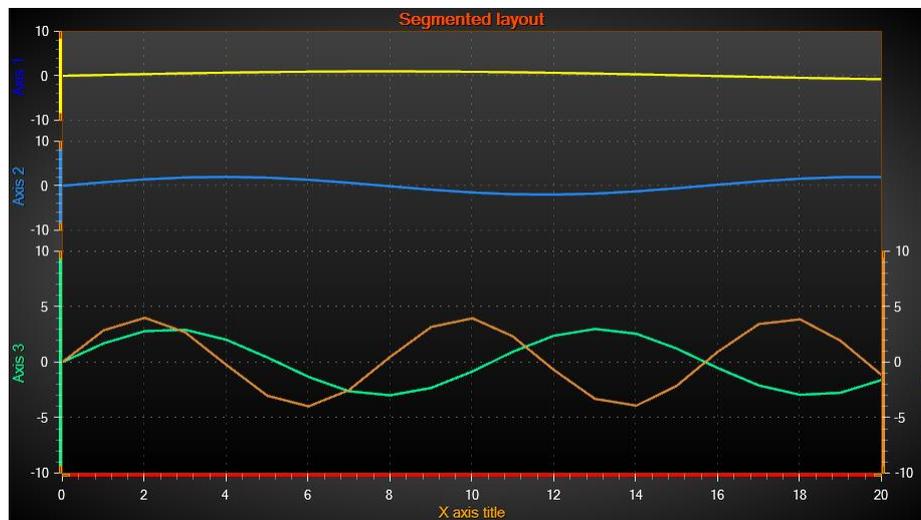


Figure 5-18. An example view of 4 Y axes in **YAxesLayout = Segmented**. First two segments have **Height = 1**, and last segment has **Height of 2.5**. **Axis1.SegmentIndex = 0**, **Axis2.SegmentIndex = 1**, **Axis3 and Axis4.SegmentIndex = 3**.

When a **Stacked** or **Segmented** view is selected, the vertical space between graph segments can be adjusted by using **ViewXY.AxisLayout.SegmentsGap** property. If there is a myriad of Y axes defined, **ViewXY.AxisLayout.AutoShrinkSegmentsGap** property should be enabled to automatically decrease the gaps. By doing so, every Y axis gets at least some vertical space to be drawn. Use **ViewXY.GetGraphSegmentInfo()** method to find out where the graph segment borders are, if in need to have segment specific user interface logic.

5.1.3 Axis grid strips

The axis grid (division) intervals can be shown over the graph background as fills. By setting **ViewXY.AxisLayout.AxisGridStrips = X**, an X axis is used to set the strips. Similarly, by setting **AxisGridStrips = Y**, Y axis is used to set the strips. **AxisGridStrips = Both** sets the strips by both X and Y axis. Setting it to **None**, no grid strips are used at all.

XGridStripAxisIndex sets the X axis that is to be used for strips. Only one X axis can be set.

YGridStripAxisIndexLayered sets the Y axis to be used for strips, when **YAxisLayout = Layered**. When **YAxisLayout = Stacked**, all Y axes have their own strips.

The strip colors can be adjusted in **GridStripColor** property of X or Y axis object.

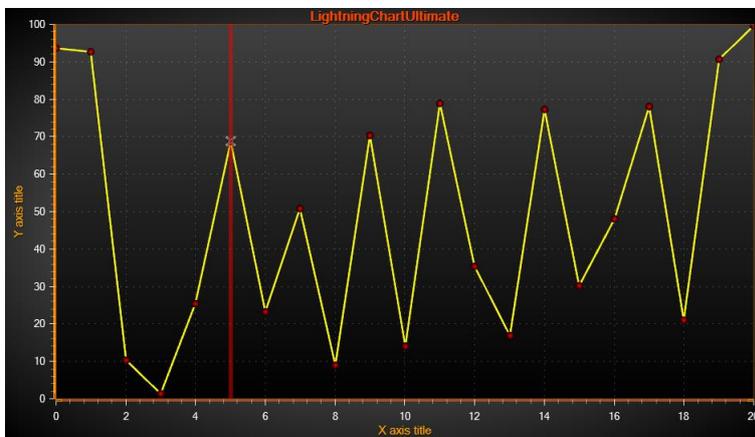


Figure 5-19. AxisGridStrips = None.

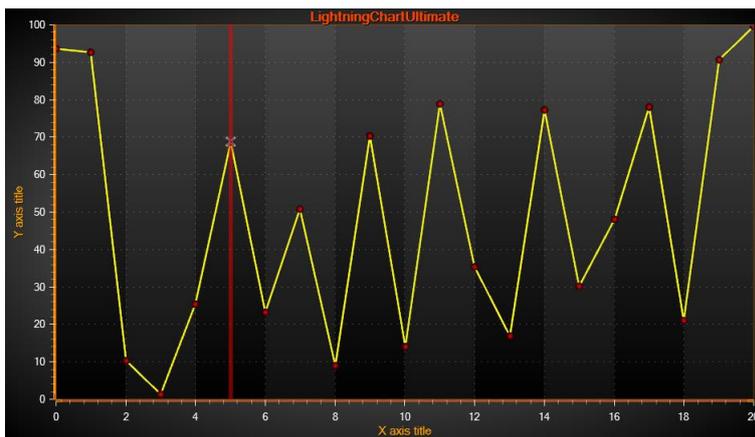


Figure 5-20. AxisGridStrips =X.

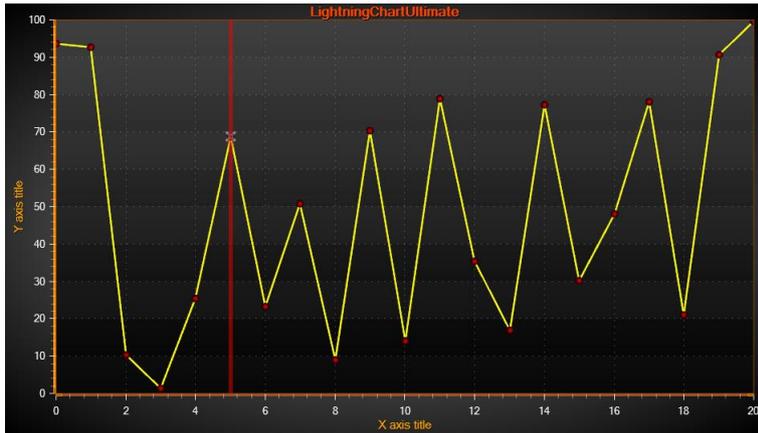


Figure 5-21. `AxisGridStrips = Y`.

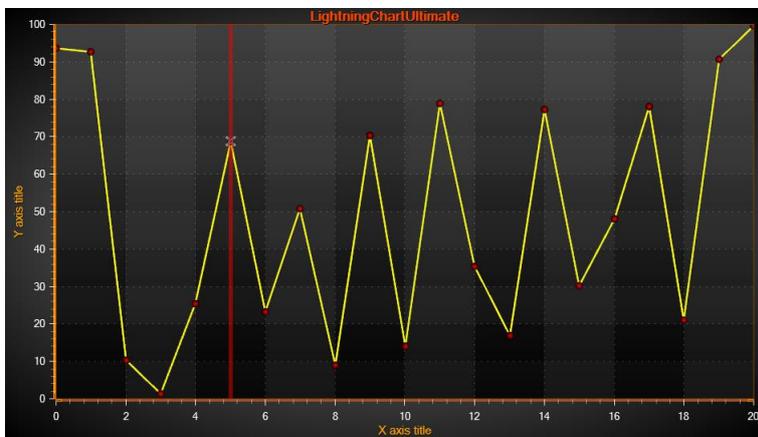


Figure 5-22. `AxisGridStrips = Both`.

5.1.4 Other `AxisLayout` options

`AutoAdjustAxisGap` sets the space between two adjacent axis areas in pixels, when `XAxisAutoPlacement` or `YAxisAutoPlacement` is enabled.

By enabling `XAxisTitleAutoPlacement` (or `YAxisTitleAutoPlacement`), the axis title distance is automatically calculated based on value labels' length, alignment options of axes and tick lines. If `XAxisTitleAutoPlacement` (or `YAxisTitleAutoPlacement`) is disabled, `Title.DistanceToAxis` of axis object property sets the distance to axis line instead.

5.2 Y axes

An unlimited count of Y axes can be defined. Add the Y axes by using **YAxes** collection property.

5.2.1 AxisY class properties

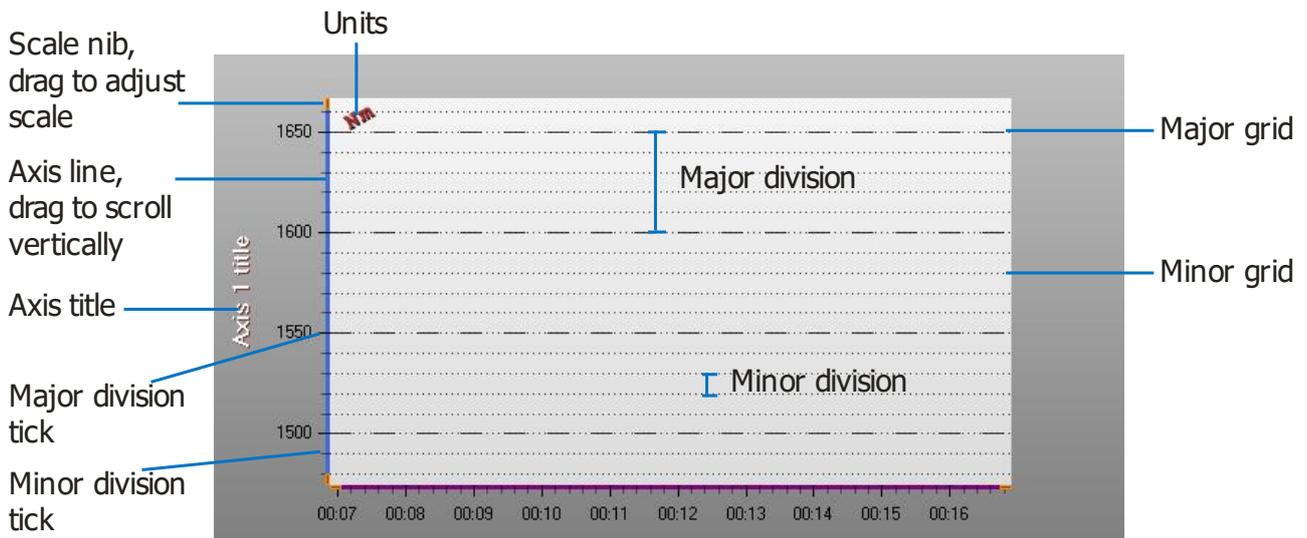


Figure 5-23. Y axis, divisions and grid.

5.2.2 Tick value labels formatting

AutoFormatLabels can be left enabled, if the count of decimals or time format representation should be calculated automatically to be suitable for visible range. To set the number formatting manually, disable **AutoFormatLabels** and use **LabelsNumberFormat** property. A standard `Double.ToString()` method parameter string applies. To set the time formatting manually, disable **AutoFormatLabels** and use **LabelsTimeFormat** property. **DateTime.ToString()** method parameters string applies, but there is also an improvement. `DateTime.ToString()` accepts only three second fractions (.fff) with `DateTime.ToString()`, but **LabelsTimeFormat** supports any count of second fractions (e.g. ".ffffff") allowing precise zoomed views. **LabelsNumberFormat** property is generally used, when **AutoFormatLabels** is enabled, to format the decimal presentation of the numbers.

5.2.3 Value type

The **ValueType** property of an axis can be set to different settings:

Number

Regular numeric format, for integer and decimal presentation. When **AutoFormatLabels** is disabled, **LabelsNumberFormat** applies.

Time

For time of day presentation. When **AutoFormatLabels** is disabled, **LabelsTimeFormat** applies.

DateTime

Date presentation, with optional time of day. When **AutoFormatLabels** is disabled, **LabelsTimeFormat** applies here as well, similarly to **Time** type.

Note! For best accuracy, it is advised to set **DateOriginYear**, **DateOriginMonth** and **DateOriginDay** just below the dates shown in the chart. Use **DateTimeToAxisValue** method to obtain axis values from a .NET **DateTime** object to be used in series data.

MapCoordsDegrees

Geographical map coordinate presentation in degrees decimals.

Example: $40.446195^{\circ} -79.948862^{\circ}$

MapCoordsDegNESW

Geographical map coordinate presentation in degrees decimals, with N, E, S, W indication.

Example: $40.446195N 79.948862W$

MapCoordsDegMinSecNESW

Geographical map coordinate presentation in degrees, arc minutes, arc seconds, with N, E, S, W indication.

Example: $40^{\circ}2'13"N 9^{\circ}58'2"W$

MapCoordsDegPadMinSecNESW

Geographical map coordinate presentation in degrees, arc minutes, arc seconds, with N, E, S, W indication. The arc minute and second values are padded with zeros, if they are < 10. It is a great way to present coordinates in Y axis, as the numbers are aligned.

Example: $40^{\circ}02'13"N 9^{\circ}58'02"W$

5.2.4 Range setting

Set the value range by giving values to **Minimum** and **Maximum** properties. **Minimum** must be less than **Maximum**. When trying to set **Minimum** > **Maximum**, or vice versa, internal limiter will limit the values near the other value. To set both values simultaneously, use **SetRange(...)** method. Passing **Minimum** > **Maximum** in **SetRange** automatically flips these values so that **Minimum** < **Maximum**.

The value range of Y axis can be scrolled directly by dragging the axis with mouse when **MouseScrolling** is enabled. **Minimum** or **Maximum** can be modified by dragging the scale nib area (end of an axis) up or down when **MouseScaling** property is enabled.

5.2.5 Restoring range

Axis has properties **RangeRevertEnabled**, **RangeRevertMaximum** and **RangeRevertMinimum**. They can be used to revert axis ranges to specific values when mouse zooming is applied from right to left. See 5.22.5 for details.

5.2.6 Divisions

The major divisions spacing can be done automatically by leaving the **AutoDivSpacing** enabled. The spacing is calculated as user-friendly as possible, depending on labels font size and **AutoDivSeparationPercent** properties. Increase the value to reduce the amount of major divisions. Minor divisions are calculated between major divisions by using **MinorDivCount** property value.

By setting **AutoDivSpacing** disabled, the division spacing can be controlled manually with **MajorDiv** property, if willing to control spacing by magnitude. Alternatively, **MajorDivCount** property can be used to control spacing by division count. **KeepDivCountOnRangeChange** property can be used to force maintaining the divisions count the same whenever the axis range is changed, since **MajorDivCount** and **MajorDiv** depend on each other.

Major division tick style can be set from **MajorDivTickStyle** property. Edit the ticks and labels orientation by using **MajorDivTickStyle.Alignment** property. The value labels are drawn next to major division ticks. Edit the minor division properties with **MinorDivTickStyle** property, respectively.

5.2.7 Grid

Horizontal grid lines are drawn on vertical positions of division ticks. Major grid for major division ticks, minor grid for minor division ticks. Use **MajorGrid** and **MinorGrid** properties to edit the appearance.

5.2.8 Custom ticks

The axis tick positions can be manually set. Set **CustomTicksEnabled** true and define the positions of the ticks in the **CustomTicks** list property.

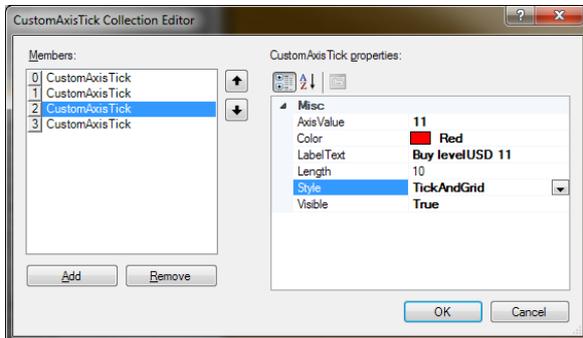


Figure 5-24. Custom tick properties

CustomAxisTicks class represents one tick, grid or both. Use **Style** to select between Tick, Grid or TickAndGrid respectively. Set the color of a tick or a grid in **Color** property. Set tick length in **Length** property. The grid line pattern follows the setting of **MajorGrid.Pattern** and **PatternScale** properties of axis.

CustomAxisTick has the **AxisValue** and **LabelText** property. When using custom ticks, disable **AutoFormatLabels** to show the custom label texts.

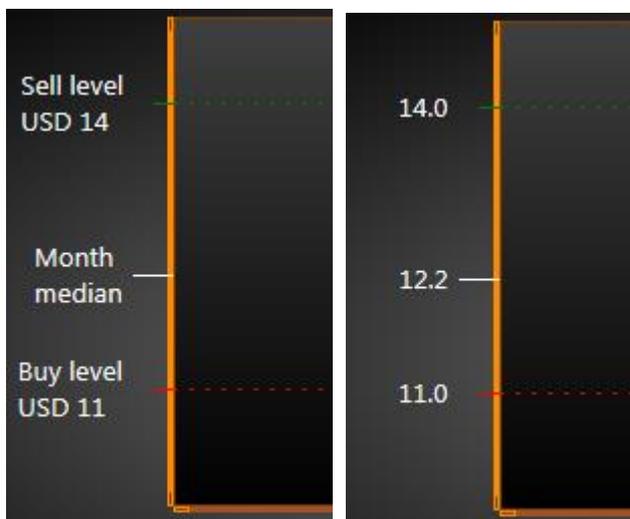


Figure 5-25. Custom ticks on Y axis. On the left, `axis.AutoFormatLabels = false`. On the right, `AutoFormatLabels = True`.

Remember to call **InvalidateCustomTicks()** after setting new ones in code.

Minor ticks or grids are not shown when **CustomAxisTicksEnabled** is true. Set arbitrary minor ticks or grids, just add **CustomAxisTicks** in the **CustomTicks** collection with different colors or line lengths.

5.2.9 Reversed X and Y axis

X and Y axis can be shown reversed, so that minimum value is above/after than the maximum value. Handy feature when, for example, visually negating polarity of the series data assigned to the Y axis.

5.2.10 Logarithmic axes

Set **ScaleType** to **Logarithmic** to use a logarithmic presentation. Set the logarithm base value with **LogBase** property. The chart can also show logarithmic values between 0...1. Use **LogZeroClamp** to set the minimum value in the axis. To use typical minimum value of a log axis, set 1. To use values below zero, set a proper small positive value, like 1.0E-20, suitable for the used data. To use special formatting for tick labels, set **LogLabelsType**.

5.2.10.1 Exponential presentation for 10 base

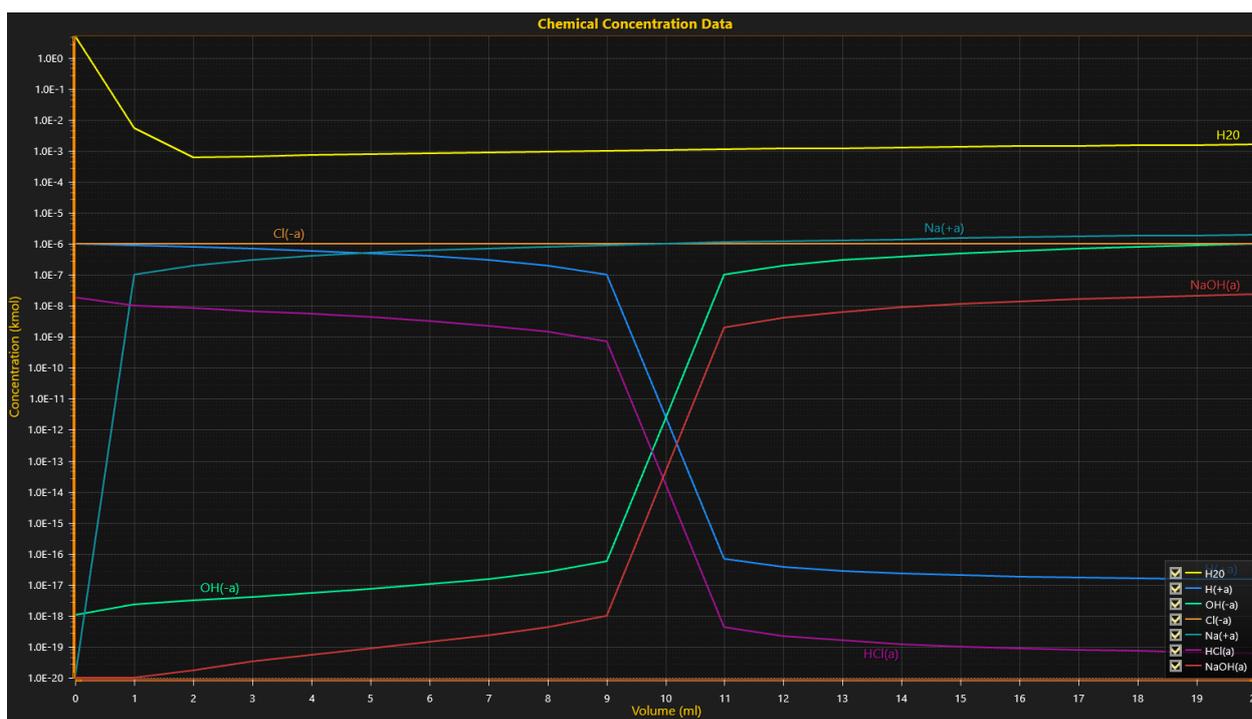


Figure 5-26. Logarithmic Y axis with values near zero. LogZeroClamp is set to 1.0E-20. LogBase is set to 10, LogLabelsType is set to Log10Exponential, to show the values in 1.0E presentation.

5.2.10.2 Natural logarithm



Figure 5-27. Natural logarithm view. LogBase is set to Math.E LogLabelsType is set to LogE_MultiplesOfNeper.

5.2.11 Converting between axis values and screen coordinates

Axes have methods to convert axis values (data point values) to screen coordinates and screen coordinates to axis values. Use **ValueToCoord** method to convert an axis value to a screen coordinate, and **CoordToValue** to convert a screen coordinate to an axis value. Set **UseDIP = False**, if pixels are preferred, not Device independent pixels (DIPs).

```
float screenCoordinate = _chart.ViewXY.XAxes[0].ValueToCoord(axisValue);
```

ValueToCoord and **CoordToValue** methods are available after the chart has got its final size. For example, subscribe to **chart.afterRendering** event to ensure the chart has been fully rendered.

To convert multiple values or coordinates at once, use **ValuesToCoords** and **CoordsToValues** methods. They take/return axis values as double arrays (integer array for X axis **CoordToValue**) and screen coordinates as float arrays.

5.2.12 MiniScale

A miniature X and Y axis substitute. In some applications this kind of scale presentation is preferred for quick visual overview of data magnitude, or alternatively, when there's no space for actual axes. Mini scales are not visible by default, and they cannot be used together with logarithmic axes. **MiniScale** is a sub-property of Y axis class. However, the X dimension of a **MiniScale** is always bound to first X axis (**XAxes[0]**). Set the visible units by modifying **Units.Text** property of X and Y axis.



Figure 5-28. MiniScale in the bottom-right corner of the graph.

5.3 X axis

X axis divisions and grid settings are equal to Y axes settings. The axis can be scrolled by mouse and the minimum and maximum value can be set by dragging the scale nibs.

5.3.1 Real-time monitoring scrolling

When making a real-time monitoring solution, the X axis must be scrolled to correctly show the current monitoring position, which usually is the time stamp of latest signal point. Set the latest time stamp to **ScrollPosition** property after the new signal points have been set to a series. LightningChart has several scrolling modes, selected using **ScrollMode** property.

5.3.1.1 None

No scrolling is applied when setting a value to **ScrollPosition**. This is often the selection to use when using the chart in other applications than real-time monitoring.

5.3.1.2 Stepping

When collected data reaches the end of the X axis, the axis with all series data is shifted left by a stepping interval. This shift is executed everytime the X axis end is reached. **SteppingInterval** property is defined as value range.

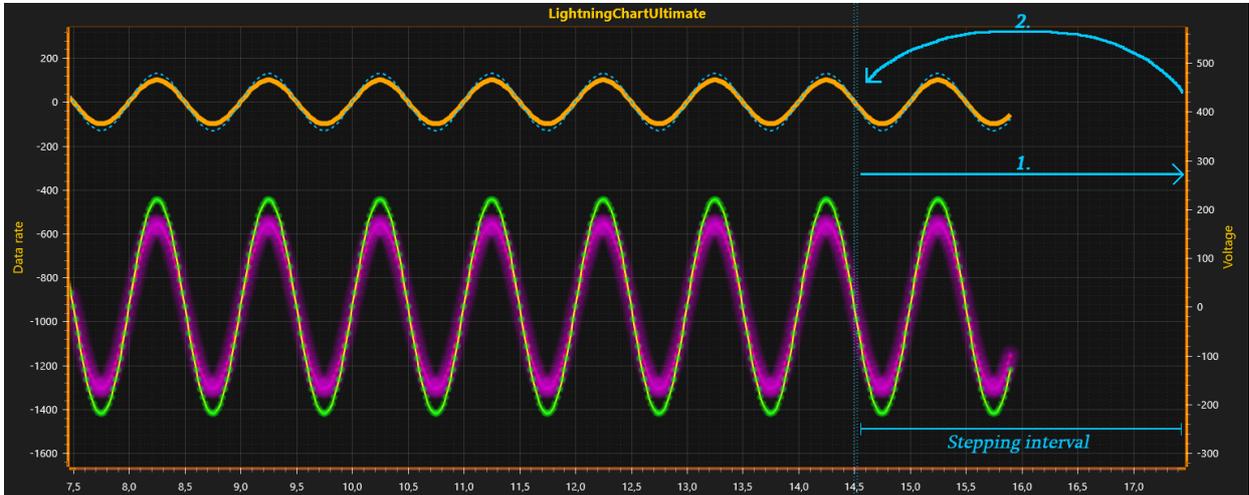


Figure 5-29. X axis scroll mode: stepping

5.3.1.3 Scrolling

The X axis is kept stationary until scrolling gap has been reached, after which the X axis with all series is continuously shifted left. If the scrolling should take effect when the scroll position reaches the end of X axis, set **ScrollingGap** to 0. **ScrollingGap** property is defined as percents of graph width.

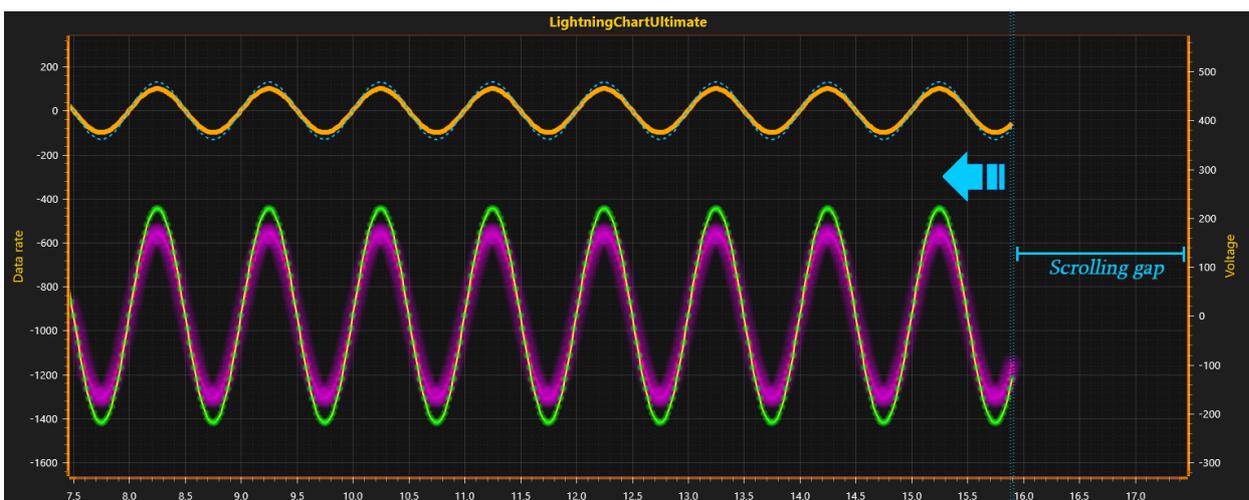


Figure 5-30. X axis scroll mode: scrolling

Waveform stability during scrolling

LightningChart supports incremental rendering data construction of real-time signal, when using **series.AddPoints()**, or **AddValues()** or **AddSamples()**. In short, it means that rendering data is calculated only from the new part of data and combined with the existing rendering data.

PointLineSeries, **SampleDataSeries**, **AreaSeries** and **HighLowSeries** have a specific property for **ScrollMode = Scrolling**, which effects the visual stability of scrolled series maintaining the waveform quality. The property is called **ScrollingStabilizing**.

When it's **enabled**, floating point coordinates are rounded to nearest integer coordinate, which results into a visually stable, non-fluctuating waveform. In most cases, this is the best approach. It may, however, distort the phase info slightly when rounding the coordinates.

When **ScrollingStabilizing** is disabled, data rendering uses floating point coordinates which appear as slightly fluctuating waveform when GPU decides the pixel coordinate. This gives better visual quality especially when displaying sine data where there's a transition going up and down nearly every other pixel.

To use incremental rendering data construction, add new points as follows

```
chart.BeginUpdate();
series.AddPoints(array, false);
xAxis.ScrollPosition = latestXValue;
chart.EndUpdate();
```

Full refresh of rendering data can be made any time with **InvalidateData()** call of series.

```
chart.BeginUpdate();
series.AddPoints(array, false);
series.InvalidateData();
xAxis.ScrollPosition = latestXValue;
chart.EndUpdate();
```

	Performance	Stability	Phase
series.AddPoints(), ScrollStabilizing disabled	Perfect	Impaired	Good
series.AddPoints(), ScrollStabilizing enabled	Perfect	Best	Slightly impaired
series.AddPoints(), InvalidateData()	Impaired	Impaired	Perfect

Table 5-1. Waveform stability during scrolling

5.3.1.4 Sweeping

Sweeping mode gives probably the most user-friendly real-time monitoring view. Sweeping uses two X axes. The first axis is collected full after which a sweeping gap appears. The second X axis is then swept over the first one. Both X axes show their own value labels. **SweepingGap** property is defined as percents of graph width.

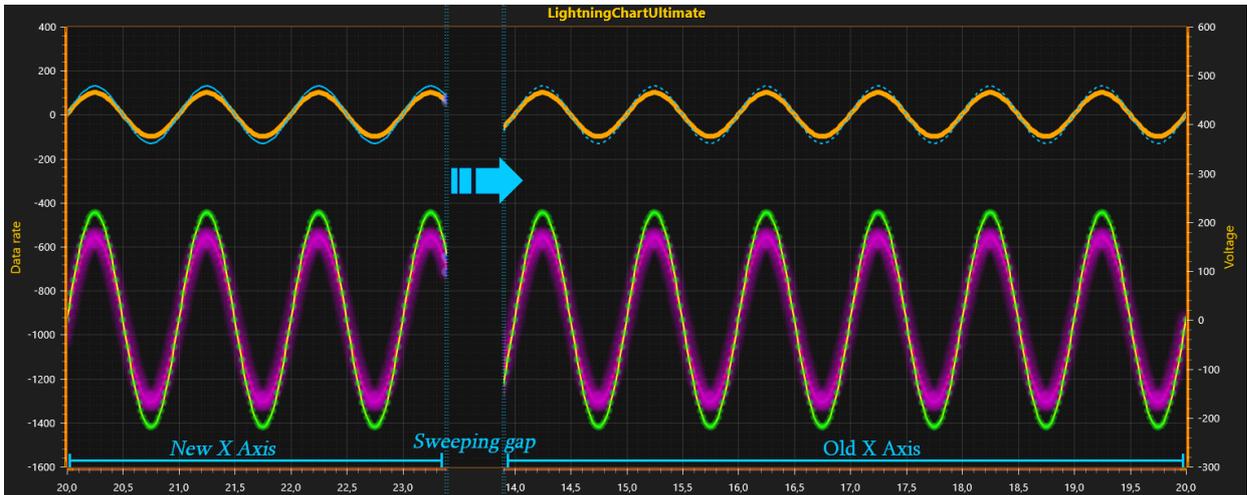


Figure 5-31. X axis scroll mode: sweeping

5.3.1.5 Triggering

The X axis position is determined by a series value exceeding or falling below a trigger level. Use **Triggering** property to set the triggering options. Triggering can be set active by enabling **Triggering.TriggeringActive** property.

One series has to be set as a triggering series. Accepted triggering series types are **PointLineSeries** and **SampleDataSeries**. Set the triggering Y level with **Triggering.TriggerLevel**. Use **Triggering.TriggeringXPosition** to order where the level triggered point will be drawn horizontally, as percents of graph width.

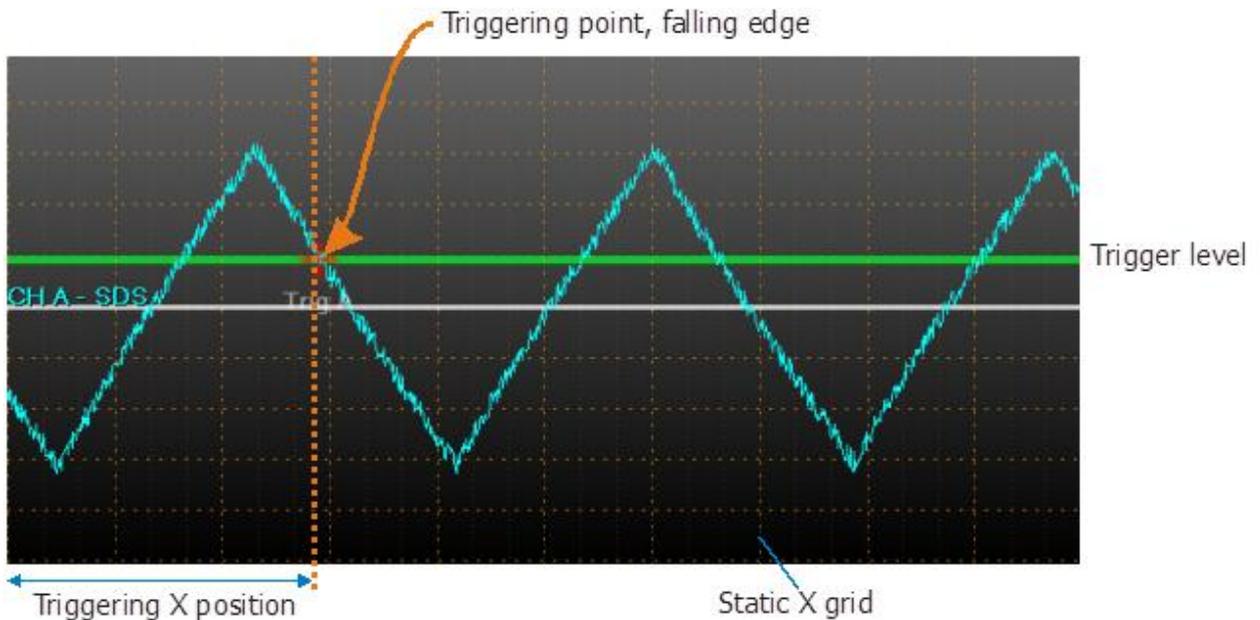


Figure 5-32. X axis scroll mode: Triggered with static X grid.

When using a triggered X-axis scroll position, it usually is not suitable to show the regular X axis with values and grid because of jumping from place to another based on the incoming series data.

- **Approach 1:** Use static X grid. Hide the regular X axis objects by setting `XAxis.Visible = false` (or `LabelsVisible = false`, `MajorGrid.Visible = false` and `MinorGrid.Visible = false`). Then, show the static X grid by setting `Triggering.StaticMajorXGridOptions` and `Triggering.StaticMinorXGridOptions`.
- **Approach 2:** Create another X axis, with preferred scale, and set it to ViewXY collection. Don't assign the second XAxis for the series.

For scale indication, use Y axis **MiniScale** or define an **Annotation** object (see chapter 5.20) to show range like "200 ms/div".

5.3.2 Scale breaks

Starting from version 8, X axes support **Scale breaks**. **Scale breaks** allow excluding specific X ranges, e.g. inactive trading hours/dates or machinery off-production hours. All the series, that have been assigned to the specified X axis, are clipped, including axis and labels themselves.

There are limitations of when **Scale breaks** can be used: `ScrollMode` must be set to `'None'` and `ScaleType` to `'Linear'`.

Insert the *ScaleBreak* objects in *ScaleBreaks* collection of X axis.

▼ Misc	
Begin	223669800
DiagonalLineSpacing	10
Enabled	True
End	223727400
> Fill	
Gap	10
> LineStyle	
Style	DiagonalLineUp

Figure 5-33. ScaleBreak properties.

Specify the range of the break with *Begin* and *End*. They are given as axis values, not DateTimes. Use *axis.DateTimeToAxisValue* method to convert them if using DateTimes.

Gap width can be adjusted with *Gap*, also 0 is accepted if no gap should be visible. Gap appearance can be configured with *Style*.

- With *Style* = 'Fill', adjust the fill with *Fill* property.
- With *Style* = 'DiagonalLineUp' or 'DiagonalLineDown', adjust the appearance with *DiagonalLineSpacing* and *LineStyle* properties.

By setting *Enabled* = *False*, the break is not effective.

PointLineSeries, *AreaSeries* and *HighLowSeries* have *ContinuousOverScaleBreak* property. By enabling it, a connecting line will be rendered over the gap.



Figure 5-34. Original trading data, Monday to Friday, 10 AM – 6 PM. Scale breaks haven't been applied. Majority of the time range doesn't have data as stock exchange has been closed making it harder to see the essential info. PointLineSeries jumping from Close-to-Close values.

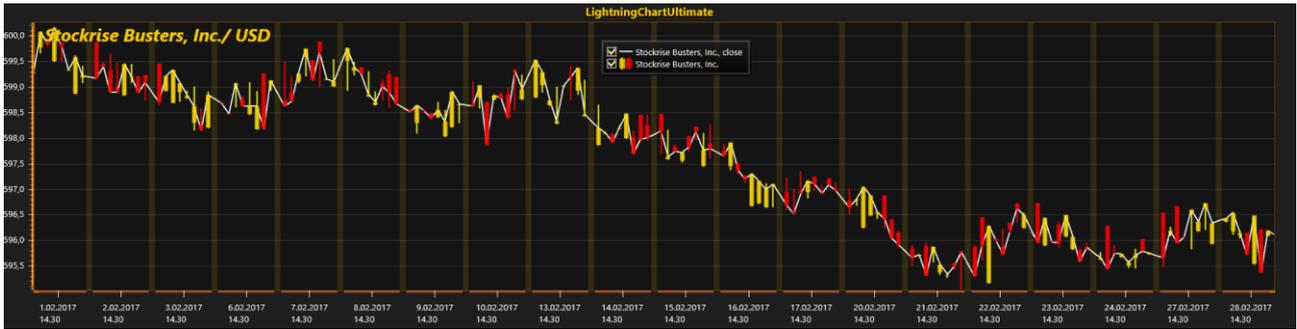


Figure 5-35. Scale breaks applied to exclude non-active trading hours. More screen space is available for essential data. Style = Fill, Gap = 10. PointLineSeries jumping from Close-to-Close values, PointLineSeries.ContinuousOverScaleBreak = True.

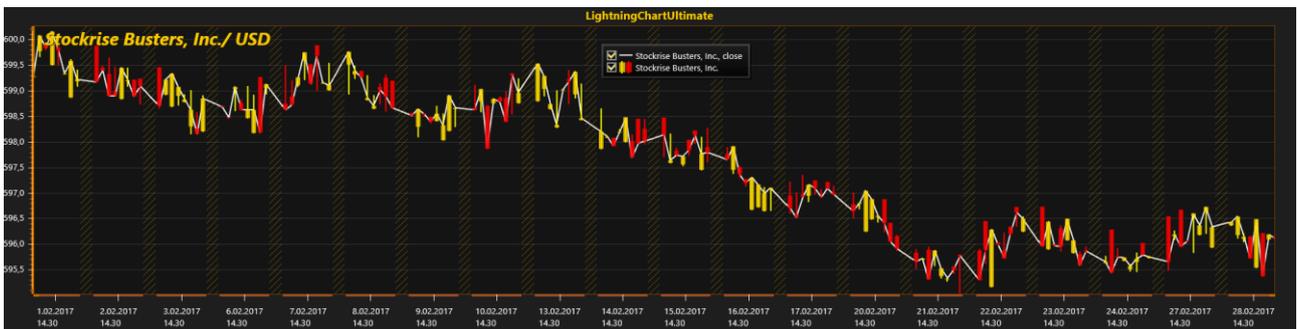


Figure 5-36. Scale breaks applied, during non-active trading hours. Style = DiagonalLinesUp, Gap = 20. PointLineSeries.ContinuousOverScaleBreak = True.

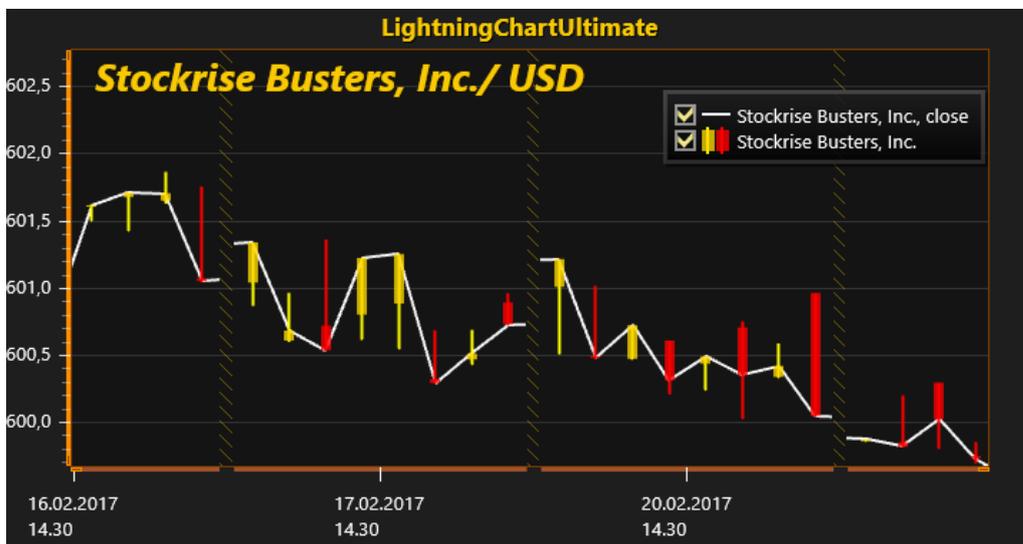


Figure 5-37. PointLineSeries.ContinuousOverScaleBreak = False. The lines are not connected from previous point to next point over the gap. Instead, they continue to their original direction as if no scale break has been defined.

5.4 Margins

Margins are empty spaces around the graph area. All the contents of the view are fitted inside the margins except for annotations, legend boxes and the chart title.

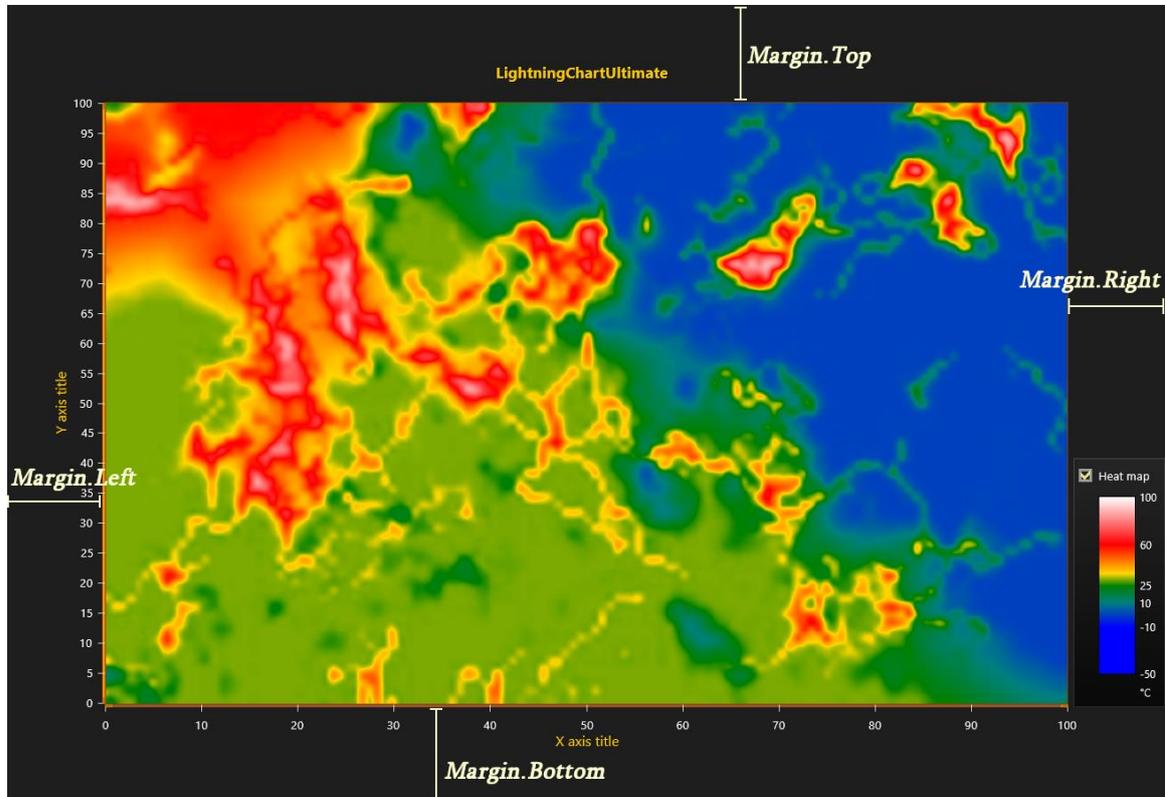


Figure 5-38. Margins surrounding the graph area. Content is fitted inside the margin area. Chart title and legend box can be positioned on margins.

When **AutoAdjustMargins** is **enabled**, the graph size is adjusted so that there is enough space for all the axes and chart title. When it is **disabled**, **ViewXY.Margins** property applies allowing setting margins manually.

By default, a customizable border rectangle, **Border**, is drawn around the graph area in the location of margins. It can be turned off by setting **Border.Visible = False**. The color of the **Border** can also be changed via **Color** property. Furthermore, **Border** can also be rendered behind the series by setting **RenderBehindSeries** to **True**.

During the run time, the margin rectangle in pixels can be retrieved by calling **ViewXY.GetMarginsRect** method, which applies to both automatic and manual margins. It is useful when needing to do screen-coordinate based computation or object placement.

ViewXY.MarginsChanged event can be set to trigger when a margin rectangle has been changed because of for example resizing it.

5.5 ViewXY series, general

ViewXY's series allow data visualization in different ways and formats. All series are bound to axis value ranges. Also, the series must be bound to one Y axis. Series have **AssignXAxisIndex** and **AssignYAxisIndex** property for assigning the X and Y axis. In code, assign the X and Y axis with series constructor parameter, alternatively.

5.6 Point line series

PointLineSeries - for variable interval progressing data

FAST TO RENDER

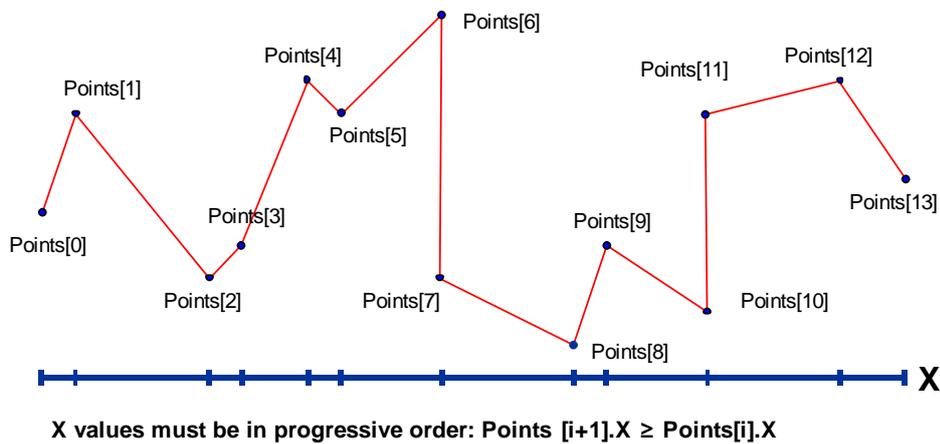


Figure 5-39. Overview of PointLineSeries

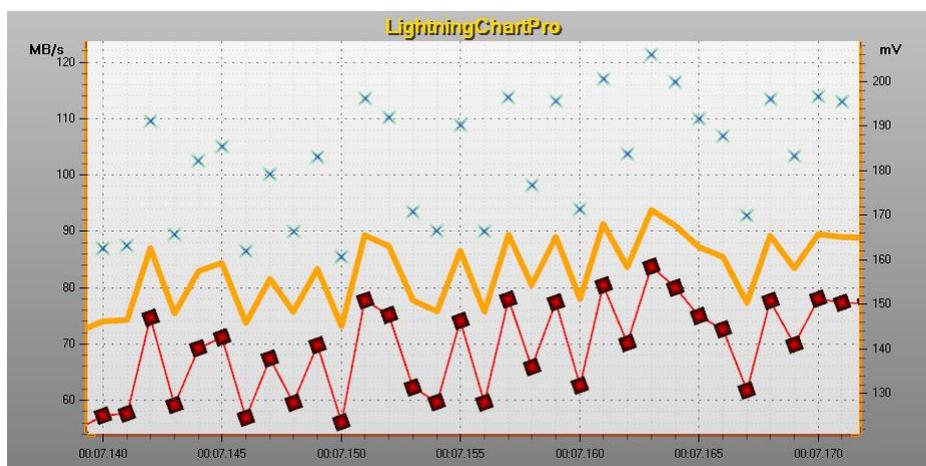


Figure 5-40. Some point line series.

A **PointLineSeries** can present a simple line, points (scatter) or both as a point line. Add the series to chart by adding **PointLineSeries** objects to **PointLineSeries** list.

```
chart.ViewXY.PointLineSeries.Add(series); //Add series to the chart
```

5.6.1 Line style

Define the line style with **LineStyle** property. If the line should not be visible, set **LineVisible = false**.

5.6.2 Points style

To show the points, set **PointsVisible = true**. Alter the point style by setting **PointStyle** properties. Select the shape from many pre-defined styles from **PointStyle.Shape**. One of the shape styles is **Bitmap**, which allows drawing any bitmap image in the point location. Define the bitmap image with **BitmapImage** property. **BitmapAlphaLevel** property can be used to alter the transparency of the bitmap. Adjust the bitmap color tone by changing **BitmapImageTintColor** to some other color than white. When using pre-defined point styles, like **Circle**, **Triangle**, **Cross** etc. the drawing colors and filling styles can be defined. Note that all colors or fills are not applicable for all shape styles. Point width and height can be set and the points can be rotated as well.

5.6.3 Coloring points individually

Starting from v.7.2, the **PointLineSeries**, **FreeformPointLineSeries**, **AreaSeries** and **HighLowSeries** have **PointColor** field in the data point structures.

To enable individual point coloring, set **IndividualPointColoring** to **Color1**, **Color2**, **Color3** or **BorderColor** setting. To disable individual point coloring, set **IndividualPointColoring = Off**. The color settings correspond to that color in **PointStyle** property.

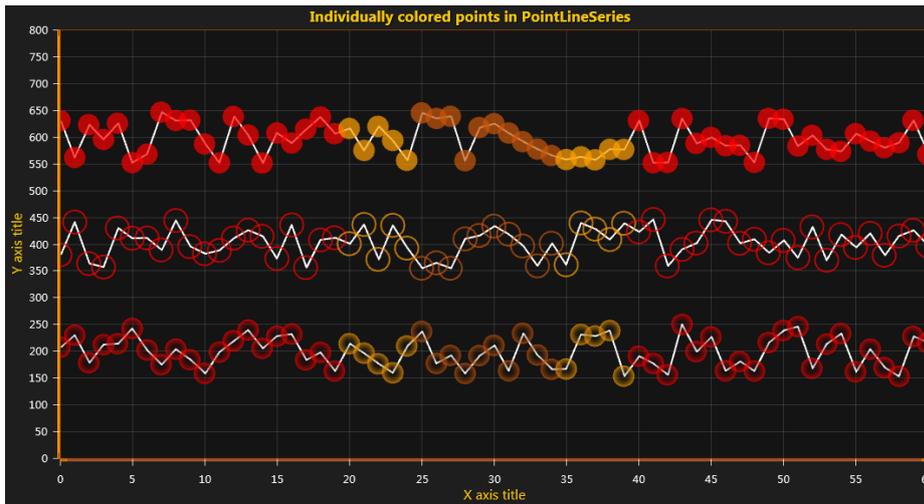


Figure 5-41. In top, `IndividualPointColoring = Color1` (solid colored point). In the middle, `IndividualPointColoring = BorderColor`. In the bottom, `IndividualPointColoring = Color2` (having gradient coloring with `Color1 = transparent`).

5.6.4 Adding points

The series points must be added in code. Use `AddPoints(SeriesPoint[], bool invalidate)` method to add points to the end of existing points.

```
chart.ViewXY.PointLineSeries[0].AddPoints(pointsArray); //Add points to the end
```

To set whole series data at once, and overwrite old points, assign the new point array directly:

```
chart.ViewXY.PointLineSeries[0].Points = pointsArray; //Assign the points array
```

Note! The `PointLineSeries` points X values must be in ascending order. If they have to be otherwise ordered, use `FreeformPointLineSeries` instead.

For example, definition `Points[0].X = 0, Points[1].X = 5, Points[2].X = 5, Points[3].X = 6` is valid.

But `Points[0].X = 2, Points[1].X = 1, Points[2].X = 6, Points[3].X = 7` is **not** a **valid** value array for `PointLineSeries`.

5.6.5 Adding points, alternative way

Points can also be added in X and Y values arrays, which turns to be more convenient way in many applications.

```
chart.ViewXY.PointLineSeries[0].AddPoints(xValuesArray, yValuesArray, false);
```

To set whole series data at once, and overwrite old points, assign the X and Y values arrays directly (applicable in WinForms and WPF Non-bindable APIs).

```
chart.ViewXY.PointLineSeries[0].SetValues(xValuesArray, yValuesArray);
```

5.7 Sample data series

SampleDataSeries - for fixed interval progressing data

VERY FAST TO RENDER

Just Y values are stored in **SamplesSingle** or **SamplesDouble** array

=> Very compact memory footprint

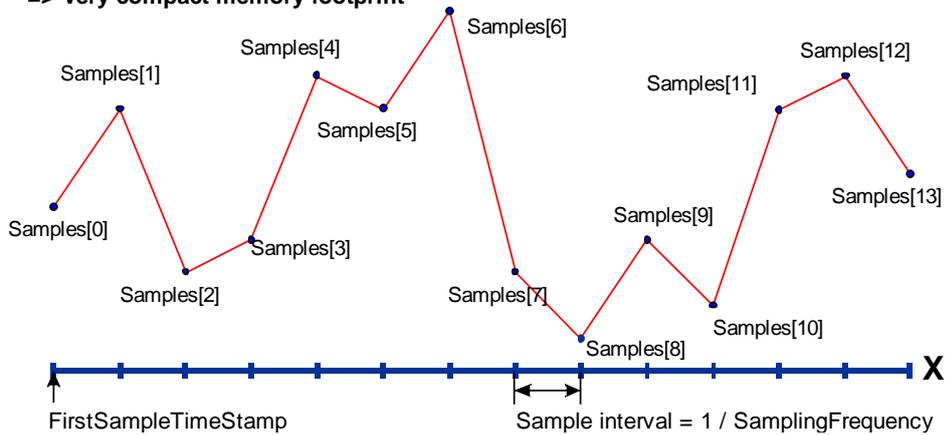


Figure 5-42. Overview of SampleDataSeries

Add the series to chart by adding **SampleDataSeries** objects to **SampleDataSeries** list.

```
chart.ViewXY.SampleDataSeries.Add(sampleDataSeries); //Add a SampleDataSeries to the chart
```

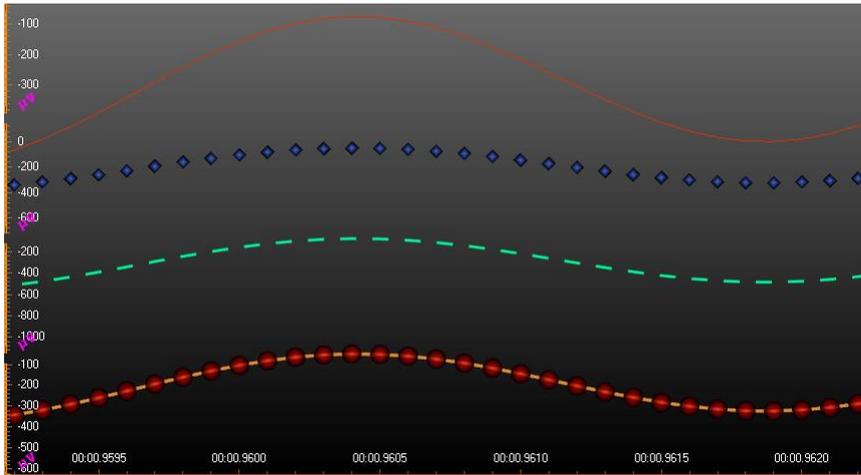


Figure 5-43. Some sample data series.

SampleDataSeries is the line series used for presenting sampled signal data (discrete signal data). This is generally used in real-time DSP applications. Visually, it is similar to **PointLineSeries**, so all line and point formatting options apply. As **SampleDataSeries** has a fixed sample interval, there's no need to reserve memory to store point X values.

Note! **SampleDataSeries** does not resample or down-sample the given data. All given data values are retained in the **SamplesSingle** or **SamplesDouble** arrays. LightningChart does not reduce the quality of the data or lose peaks or accuracy of the data.

5.7.1 Y precision

The **SampleDataSeries** supports single and double precision sample Y values. Using single precision values is recommended when keeping the memory reserving as low as possible. Select the sample format with **SampleFormat** property.

Use the series **SamplingFrequency** (1 / sample interval) to set the fixed sample interval. To set the X value (time stamp) where the samples begin, set **FirstSampleTimeStamp** property.

5.7.2 Adding points

The samples must be added in code. Use **AddSamples** method to add samples to the end of existing samples.

```
chart.ViewXY.SampleDataSeries[0].AddSamples(samplesArray, false); //Add
samples to the end
```

To set whole series data at once, and overwrite old samples, assign the new samples array directly:

If **SampleFormat** is **SingleFloat**

```
chart.ViewXY.SampleDataSeries[0].SamplesSingle = samplesSingleArray;
```

Or if **SampleFormat** is **DoubleFloat**

```
chart.ViewXY.SampleDataSeries[0].SamplesDouble = samplesDoubleArray;
```

5.8 Freeform point line series

FreeformPointLineSeries - for arbitrary data

HEAVY TO RENDER WHEN POINT COUNT IS VERY HIGH

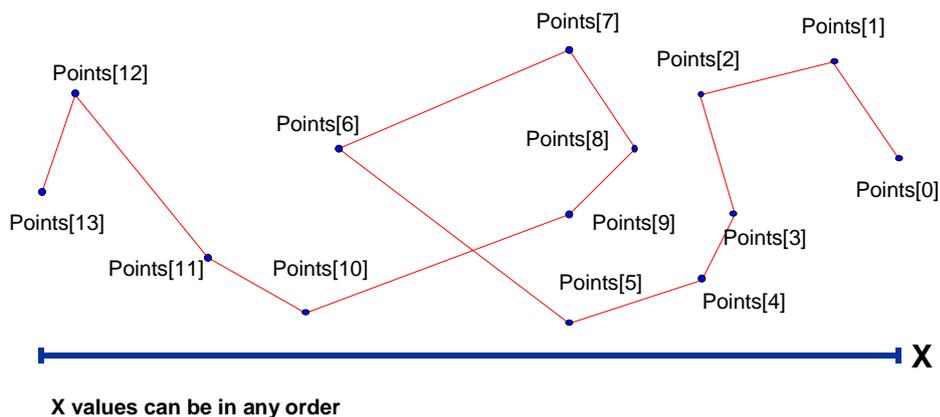


Figure 5-44. Overview of FreeformPointLineSeries

A **FreeformPointLineSeries** can present a simple line, points (scatter) or both as a point line. **FreeformPointLineSerie** allows drawing line point to any direction from previous point. All line and point formatting options from **PointLineSeries** apply. Add the series to chart by adding **FreeformPointLineSeries** objects to **FreeformPointLineSeries** list.

```
chart.ViewXY.FreeformPointLineSeries.Add(freeformPointLineSeries); //Add a  
FreeformPointLineSeries to the chart
```

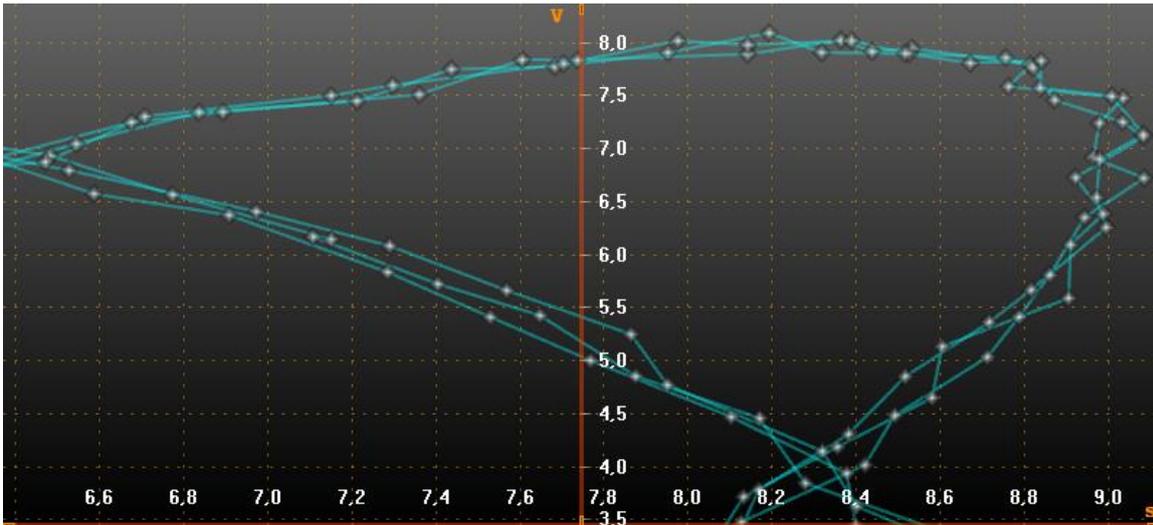


Figure 5-45. A freeform point line series

Freeform point line series line points are not automatically destroyed even if **DropOldData** is enabled and the points are scrolled out of current view. To automatically destroy old series points in real-time monitoring solution, use point count limiter. Set **PointCountLimitEnabled = true** and set the limit to **PointCountLimit** property. If limiter is enabled, the **Points** array behaves as a ring buffer after the point count limit has been reached. The oldest point from **Points** array can always be found by retrieving value from **OldestPointIndex**. If needing to read the existing data out of point count limited buffer, use the following method:

- If **OldestPointIndex** is 0, read from **Points[0]** till **Points[PointCount-1]**.
- If **OldestPointIndex** > 0, first read from **Points[OldestPointIndex]** till **Points[PointCountLimit-1]**. Then, read from **Points[0]** till **Points[OldestPointIndex-1]**.

To directly retrieve the last series point, call **GetLastPoint()** method.

5.9 Advanced line coloring of line series

The line color can be changed based on data values, or on other external logic.

5.9.1 Y-value based coloring of line and fill with value-range palette

By enabling **UsePalette** property of **SampleDataSeries**, **PointLineSeries** or **FreeformPointLineSeries**, the coloring of line is applied by the **ValueRangePalette** property. **ValueRangePalette** contains Y values and color pairs. **ValueRangePalette.Type** sets the **Gradient** or **Uniform** steps palette.

The palette coloring can be set for Y axis line too. Enable **UsePalette** property of Y axis and assign the preferred series in **PaletteSeries** property.

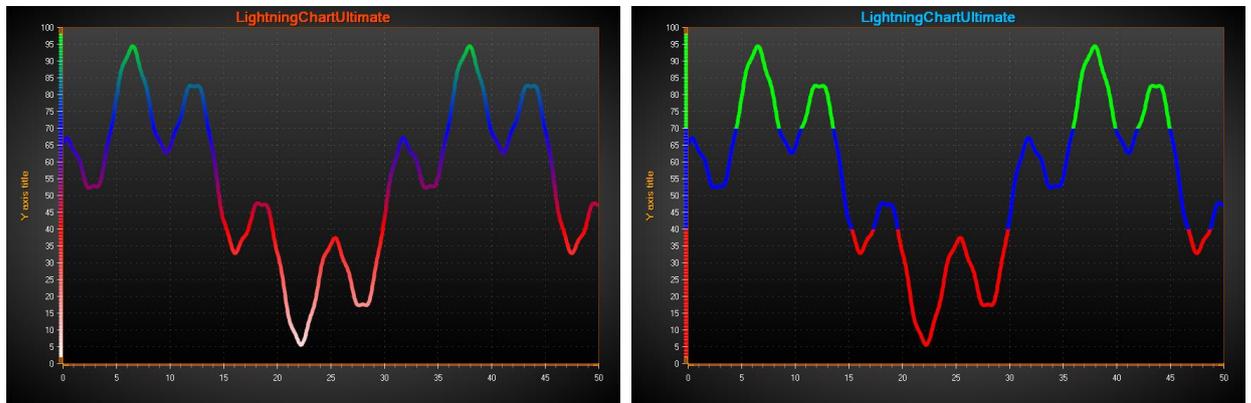


Figure 5-46. On the left, a Gradient palette is used to color the line based on Y values. On the right, a Uniform palette is used. UsePalette is enabled for Y axis as well.



Figure 5-47. Gradient palette coloring for bipolar signal data. UsePalette for Y axis is disabled.

5.9.2 Custom shaping and coloring with CustomLinePointColoringAndShaping event

Custom coloring and coordinate adjustment can be made with **CustomLinePointColoringAndShaping** event, which is called just before entering the rendering stage of the chart.

The custom coloring is available when *LineStyle.Pattern = Solid*. It also has serious limitations in vector file exports.

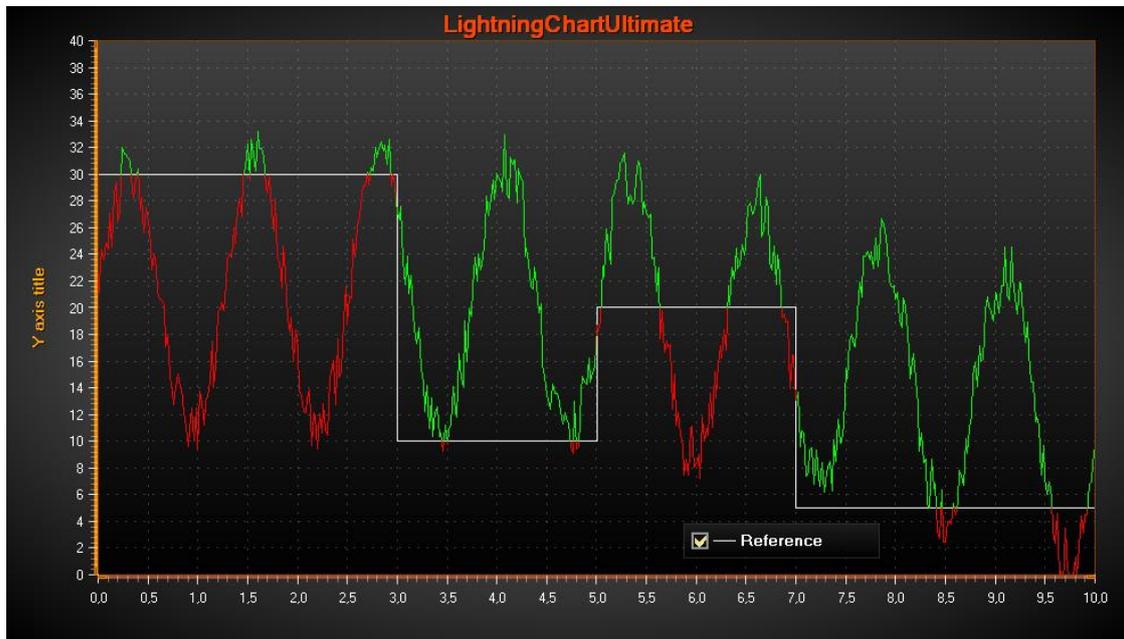


Figure 5-48. CustomLinePointColorAndShaping event handler used to change the line color by the specific changing reference level.

The event arguments have the following info:

- **CanModifyColors:** Colors modification is available.
- **Colors:** Prefilled colors array with LineStyle color. If **CanModifyColors** is true, modifications can be done either by assigning new values to prefilled colors, or by creating a new colors array. If **CanModifyColors** is false, don't fill them.
- **CanModifyCoords:** Coordinates modification is available.
- **Coords:** Pre-filled screen coordinates array. If **CanModifyCoords** is true, modifications can be done either by assigning new values to prefilled coordinates, or by creating a new coordinates array. The new array length doesn't have to be equal to prefilled one. **Ensure the length of the Coords and Colors array are equal when exiting the event handler.** If **CanModifyCoords** is false, don't fill them.
- **HasDataPointIndices:** Only applicable in *FreeformPointLineSeries*.
- **DataPointIndices:** Data point indices that were included in the coordinates and colors arrays. The chart will skip subsequent points in line construction, if their X and Y values or coordinates are equal. Using DataPointIndices info, e.g. a color can be picked for a line point from data point's **PointColor** field or external color array.
- **SweepPageIndex:** If *XAxis.ScrollMode = 'Sweeping'*, tells the page index (0 or 1).

5.10 High-low series

High-low series presents data as filled area between high and low values. Add the series to chart by adding **HighLowSeries** objects into **HighLowSeries** list.

```
//Add high-low series to the chart  
chart.ViewXY.HighLowSeries.Add(highlowSeries);
```

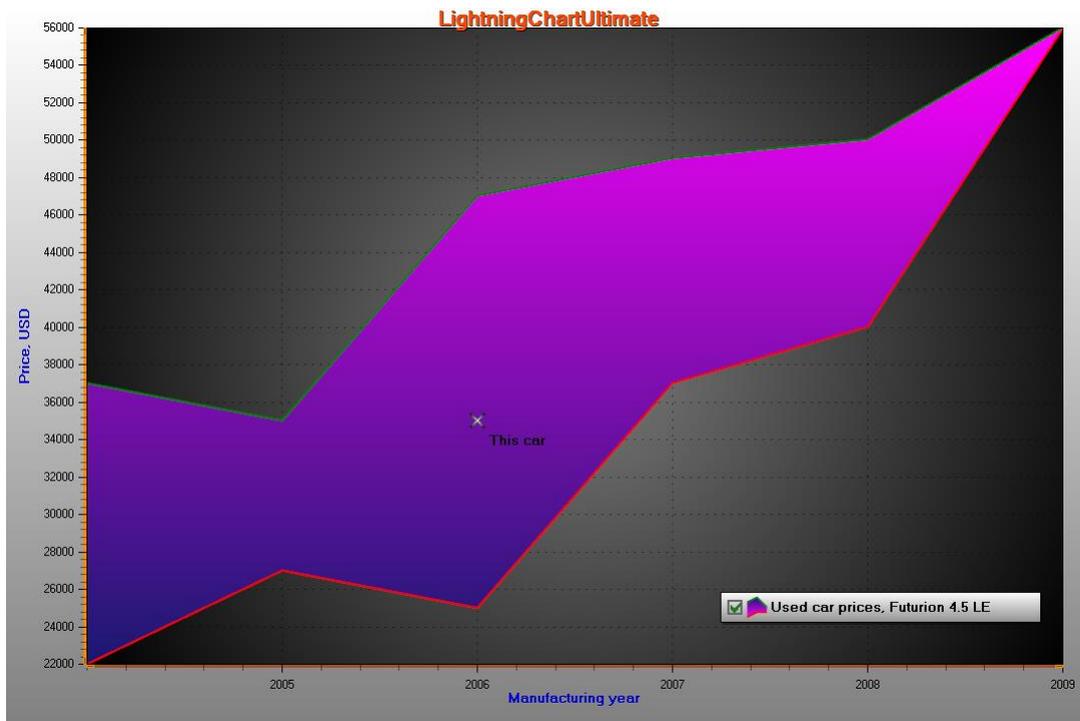


Figure 5-49. A high-low series with a marker over it.

5.10.1 Fill, line and point styles

The fill can be set with **Fill** property and its sub-properties. Define the line style with **LineStyleHigh** and **LineStyleLow** properties. If the lines should not be visible, set **LineVisibleHigh = false**, and **LineVisibleLow = false**, respectively. Define the point style with **PointStyleHigh** and **PointStyleLow** properties. If the points should not be shown, set **PointsVisibleHigh = false**, **PointsVisibleLow = false**. See chapters 5.6.1 and 5.6.2 for line and point style details. When the high value of the data is less than its Low value, reverse fill is applied in that part. Edit the reversed fill with **ReverseFill** property.

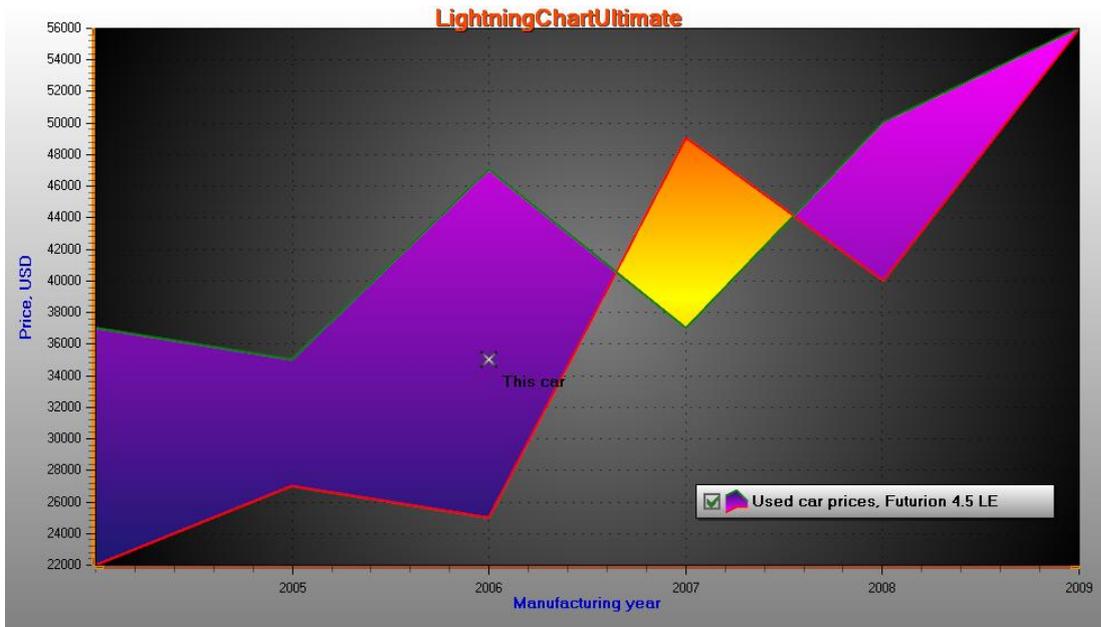


Figure 5-50. Fourth data item is given reversed: high value is < low value.

5.10.2 Limits

By enabling *UseLimits*, series shows different solid coloring above the *exceed limit* and below *deceed limit*. The regular *Fill* and *ReverseFill* apply then only for the range between the limits.

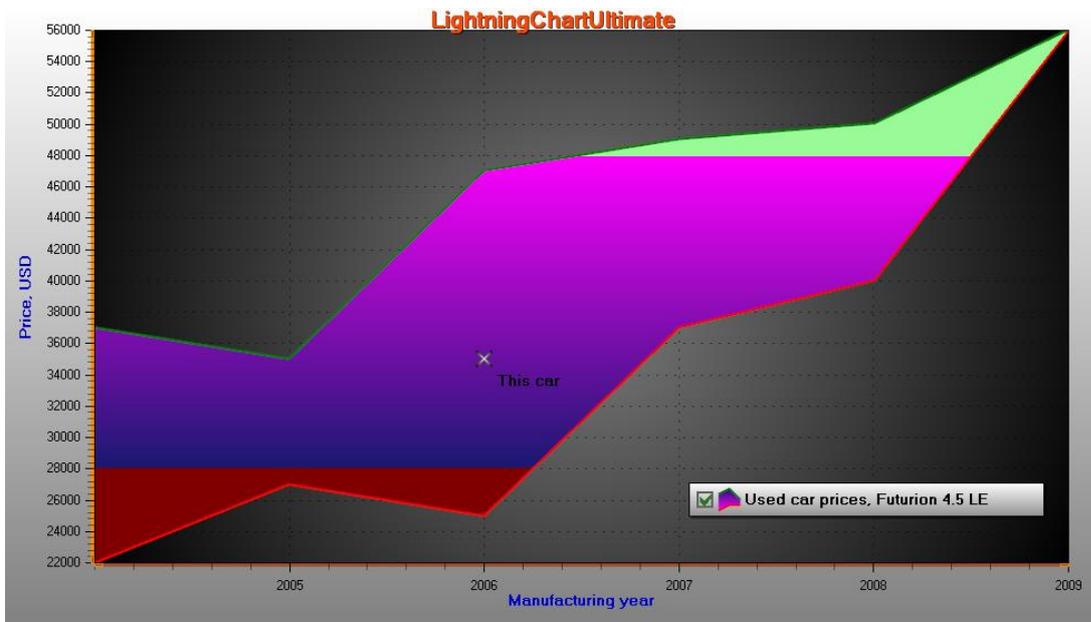


Figure 5-51. UseLimits = true, ExceedLimit = 48000 and DeceedLimit = 28000.

5.10.3 Coloring by value-range palette

By enabling *UsePalette*, the fill uses *ValueRangePalette* steps. *Uniform* and *Gradient* coloring are both supported.

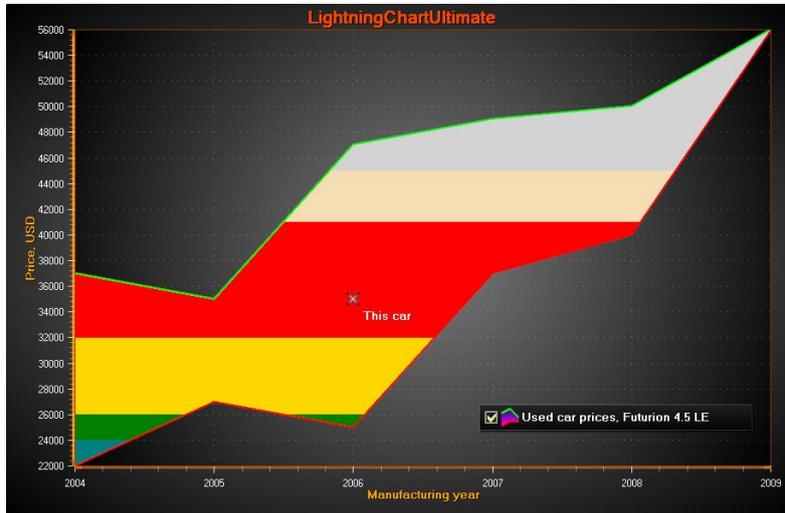


Figure 5-52. UsePalette = True, several steps defined in ValueRangePalette. Uniform coloring.

5.10.4 Adding data

The data values must be added in code. The data must be given in ascending order by X values, **Points[i+1].X ≥ Points[i].X**.

Use *AddValues(HighLowSeriesPoint[], bool invalidate)* method to add data values to the end of existing values array.

```
HighLowSeriesPoint[]dataArray = new HighLowSeriesPoint[6];
dataArray [0] = new HighLowSeriesPoint(2004, 37000, 22000);
dataArray [1] = new HighLowSeriesPoint(2005, 35000, 27000);
dataArray [2] = new HighLowSeriesPoint(2006, 47000, 25000);
dataArray [3] = new HighLowSeriesPoint(2007, 37000, 49000);
dataArray [4] = new HighLowSeriesPoint(2008, 40000, 50000);
dataArray [5] = new HighLowSeriesPoint(2009, 56000, 56000);

//Add data to the end
chart.ViewXY.HighLowSeries[0].AddValues(dataArray, true);
```

To set whole series data at once while overwriting old data, assign the new data array directly:

```
//Assign the data into points array
chart.ViewXY.HighLowSeries[0].Points = dataArray;
```

5.11 Area series

Area series presents data as filled area between base level and values. Area series is quite similar to **HighLowSeries** described in chapter 5.10, but simpler. Add the series to chart by adding **AreaSeries** objects into **AreaSeries** list.

```
chart.ViewXY.AreaSeries.Add(areaSeries); //Add area series to the chart
```

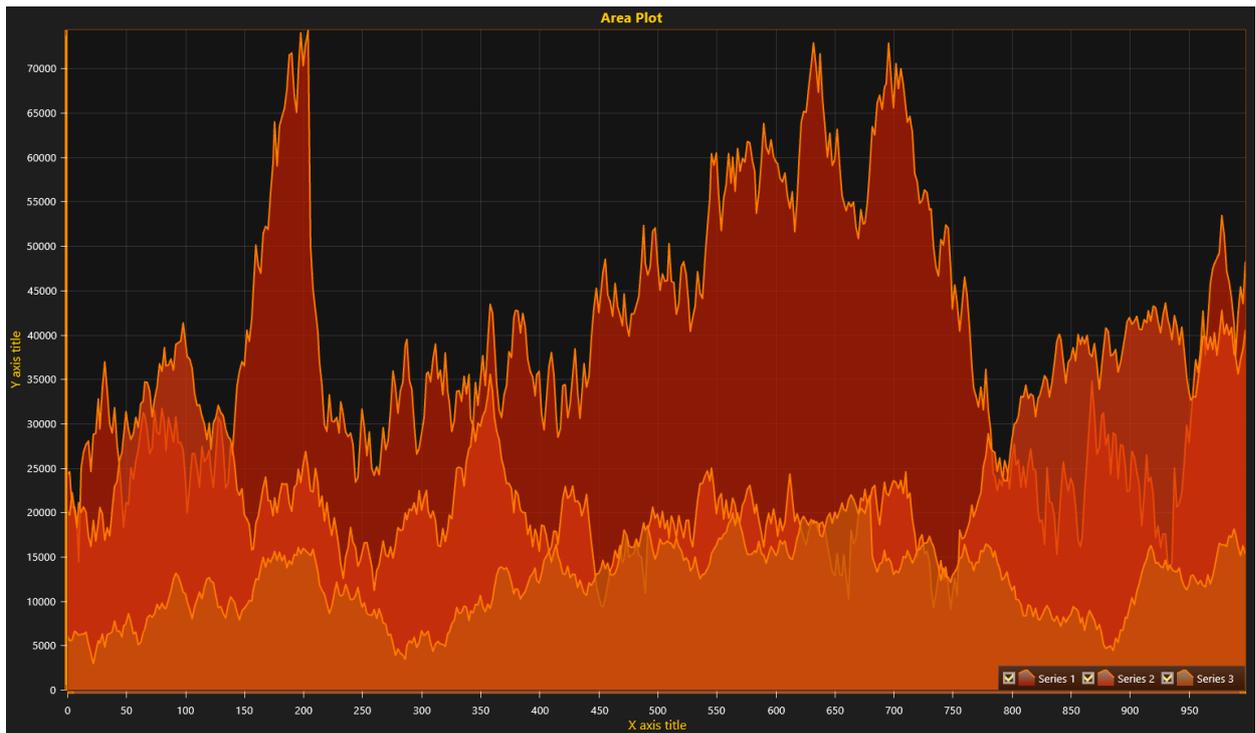


Figure 5-53. Three area series all having **BaseValue = 0**.

Set base level with **BaseValue** property. Set the preferred fill style with **Fill** property. Line style can be set with **LineStyle** property and point style with **PointStyle** property respectively. Exceed and deceed limits can be used like in **HighLowSeries**.

5.11.1 Adding data

The data values must be added in code. The data must be given in ascending order by X values, **Points[i+1].X ≥ Points[i].X**.

Use **AddValues(AreaSeriesPoint[], bool invalidate)** method to add data values to the end of existing values array.

```
AreaSeriesPoint[] dataArray = new AreaSeriesPoint[6];
dataArray [0] = new AreaSeriesPoint (2004, 37000);
dataArray [1] = new AreaSeriesPoint (2005, 35000);
dataArray [2] = new AreaSeriesPoint (2006, 47000);
dataArray [3] = new AreaSeriesPoint (2007, 37000);
dataArray [4] = new AreaSeriesPoint (2008, 40000);
dataArray [5] = new AreaSeriesPoint (2009, 56000);

//Add data to the end
chart.ViewXY.AreaSeries[0].AddValues (dataArray, true);
```

To set whole series data at once while overwriting old data, assign the new data array directly:

```
//Assign the data into points array
chart.ViewXY.AreaSeries[0].Points = dataArray;
```

5.12 Bars

Bar series allows displaying data in horizontal or vertical bars.

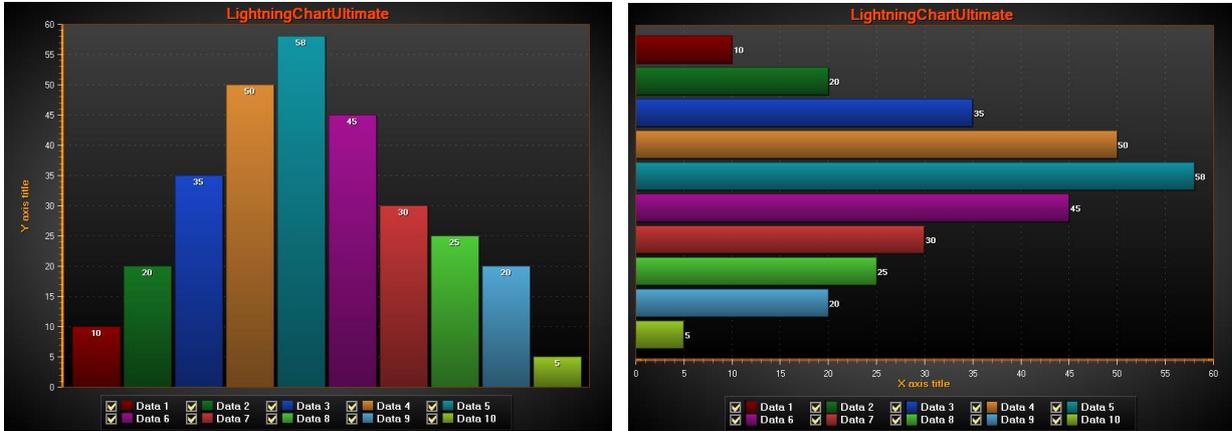


Figure 5-54. Bars series, vertical and horizontal.

Use **Values** array property to store the values of a bar series. Add values *with* **AddValue(...)** method. Update an existing value by given value index with **SetValue(...)** method. The values are of type **BarSeriesValue**, which has the following fields:

- **Value** The bar length.
- **Location** X axis location of the bar (vertical presentation) or Y axis location (horizontal presentation).
- **Text** The text that appears in the bar.

Use **LabelStyle** property of a bar series to control how the bar value label appears on the chart. The label value text is set by **AddValue(...)** or **SetValue(...)** method parameter. Various fill styles can be used by setting **Fill** property and its sub-properties.

Use **BarViewOptions** property of the chart to control how the bars are displayed. **BarView.Options.Orientation** to selects between **Horizontal** and **Vertical** bar orientation.

BarViewOptions.Grouping allows grouping the bars by value indices, by indices using width fitting or by location values. It brings values from different bar series visually together. If no grouping is wanted, use **BarViewOptions.Grouping.ByLocation** and set different **Location** field for every **BarSeriesValue** object. Use width fitting properties to adjust the spaces between columns and aside them. When no width fitting is used, **BarThickness** property of the bar series determines the bar width. The groups can be stacked by setting **BarViewOptions.Stacking** to **Stack** or to **StackStretchToSum**. When using **StackStretchToSum**, define the target sum by setting **StackSum** property. It is 100 by default to represent 100 %.

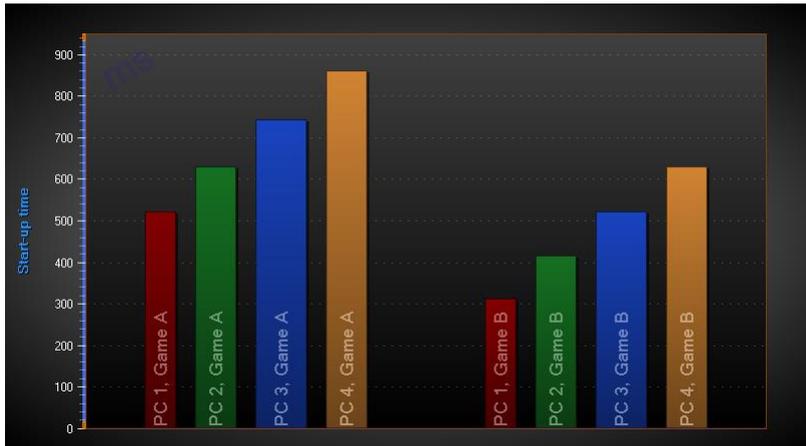


Figure 5-55. Bars series Grouping = ByIndex, Stacking = None.

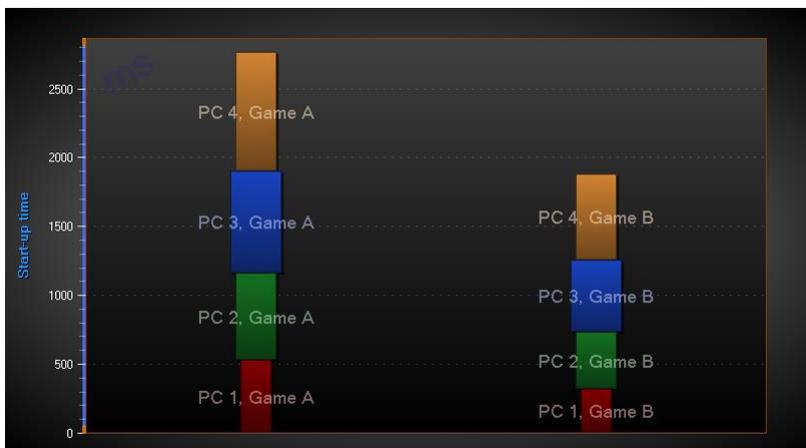


Figure 5-56. Bars series Grouping = ByIndex, Stacking = Stack.

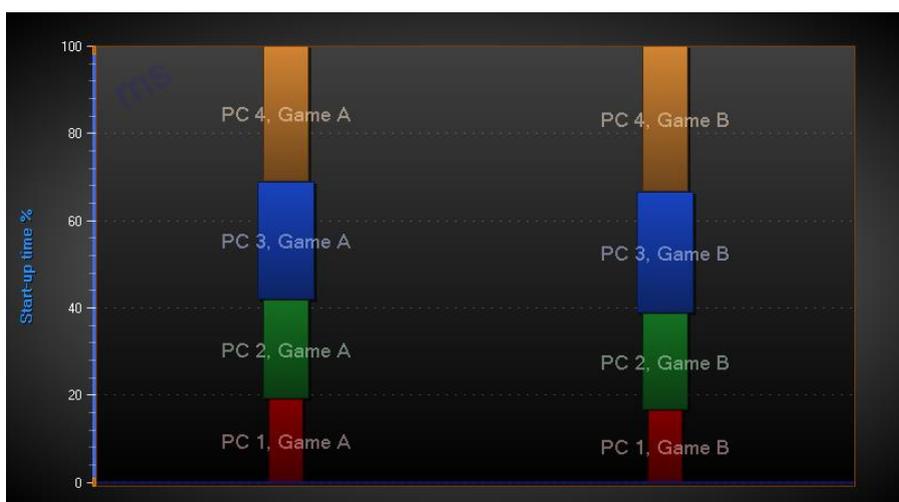


Figure 5-57. Bars series Grouping = ByIndex, Stacking = StackStretchToSum. StackSum = 100.

BaseLevel property in **BarSeries** is the series minimum value for all values and sets bar start position. In **Stacked** view, it will increase (if positive) or decrease (if negative) the size of the bar. If **StackedToSum**, the bar size is relative and calculated like **Stacked**.

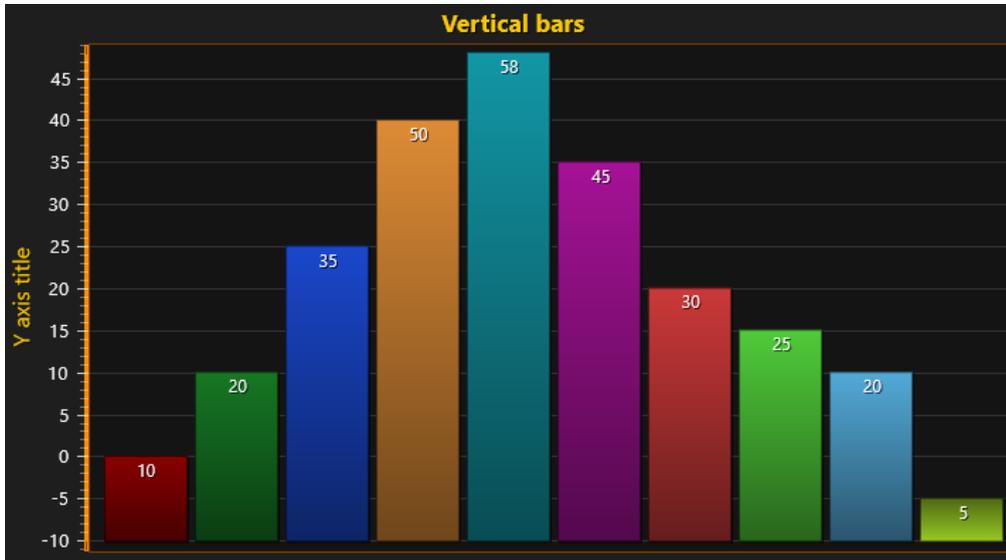


Figure 5-58. BaseLevel set to -10. Bar values are 10, 20, 35, 50, 58, 45, 30, 25, 20, 5.

5.13 Stock series

Stock series allow stock exchange data visualization in candle-stick or stock bars format. Several stock series can be added in the same chart by adding several **StockSeries** objects in **StockSeries** list property. Select the style with Style property. The options are: **Bars**, **CandleStick** and **OptimizedCandleStick**.

Set the coloring and filling options with **ColorStickDown**, **ColorStickUp**, **FillDown** and **FillUp** properties. Adjust the stick width in pixels with **StickWidth** property. Adjust the total data item width with **ItemWidth** property.

For maximum rendering performance, use **Bars** style, with **StickWidth** = 1.

OptimizedCandleStick is used by default for performance reasons, starting from v.8.4. However, **OptimizedCandleStick** has only limited set of fill effects available - it supports **Solid** and left-to-right direction **Linear** fill. **Bitmap**, **Radial**, **RadialStretched** and **Cylindrical** fills are not supported. **OptimizedCandleSticks** don't support borders of the candles - **FillBorder** is not applicable. Set **Style** = **CandleStick** for more advanced appearance options.

StockSeries can be set to render before the line series, by setting **Behind** = **True**.



Figure 5-59. One StockSeries with Style = CandleStick. Light blue line is a PointLineSeries in the background going through all Close values.



Figure 5-60. One StockSeries with Style = Bars. Line series are used for showing linear regression fit and offset of that line (2 * standard deviation). A band is used for selecting a date range for line fit.

5.13.1 Setting data to StockSeries

Create a data array and set the array items. Each item has the following fields:

Date	DateTime value (year, month, day)
Open	Opening value of the day
Close	Close value of the day
Low	The lowest value during day
High	The highest value during day
Transaction	The total trading sum
Volume	Count of shares traded

Keep the data always in ascending order by Date value (oldest date first).

```
// Create data array
StockSeriesData[] data = new StockSeriesData[] {
    new StockSeriesData(2010,09,01, 24.35, 24.76, 24.81, 23.82,
        269210, 6610451.55),
    new StockSeriesData(2010,09,02, 24.85, 24.66, 24.85,
        24.53, 216395, 5356858.225),
    new StockSeriesData(2010,09,03, 24.80, 24.84, 25.07,
        24.60, 164583, 4084950.06),
    new StockSeriesData(2010,09,06, 24.85, 25.01, 25.12,
        24.84, 118367, 2950889.31)
};
// Assign the data array to series
chart.ViewXY.StockSeries[0].DataPoints = data;
```

5.13.2 Setting X axis to date display

```
chart.ViewXY.XAxes[0].ValueType = AxisValueType.DateTime;
chart.ViewXY.XAxes[0].LabelsAngle = 90;
chart.ViewXY.XAxes[0].LabelTimeFormat =
    System.Globalization.CultureInfo.CurrentCulture.DateTimeFormat
        .ShortDatePattern;
chart.ViewXY.XAxes[0].MajorDiv = 24 * 60 * 60; //major div is one day in seconds
chart.ViewXY.XAxes[0].AutoFormatLabels = false;

//Set datetime origin
chart.ViewXY.XAxes[0].DateOriginYear = data[0].Date.Year;
chart.ViewXY.XAxes[0].DateOriginMonth = data[0].Date.Month;
chart.ViewXY.XAxes[0].DateOriginDay = data[0].Date.Day;
```

Set the X axis range suitable for data:

```
// x-axis stretched half a day at both ends. Use first and last date value
chart.ViewXY.XAxes[0].SetRange(
    chart.ViewXY.XAxes[0].DateTimeToAxisValue(data[0].Date) - 12 * 60 * 60,
    chart.ViewXY.XAxes[0].DateTimeToAxisValue(data[data.Length - 1].Date) + 12
    * 60 * 60);
```

5.13.3 Custom formatting of appearance

The *StockSeries* has *CustomStockDataAppearance* event handler, which can be used to format appearance of series data items individually, overriding the generic fill and color styles applied with properties. In the event handler, modify width and colors for specific points.



Figure 5-61. CustomStockDataAppearance used to highlight specific data items with greater width and brighter gradient colors.

5.13.4 Applying Scale breaks

To cut off non-trading hours and days, see 5.3.2.

5.14 PolygonSeries

PolygonSeries renders a fill and a borderline, by given border path.

Set the filling preferences in **Fill** property. Use **Border** property of **PolygonSeries** to set the border line style.

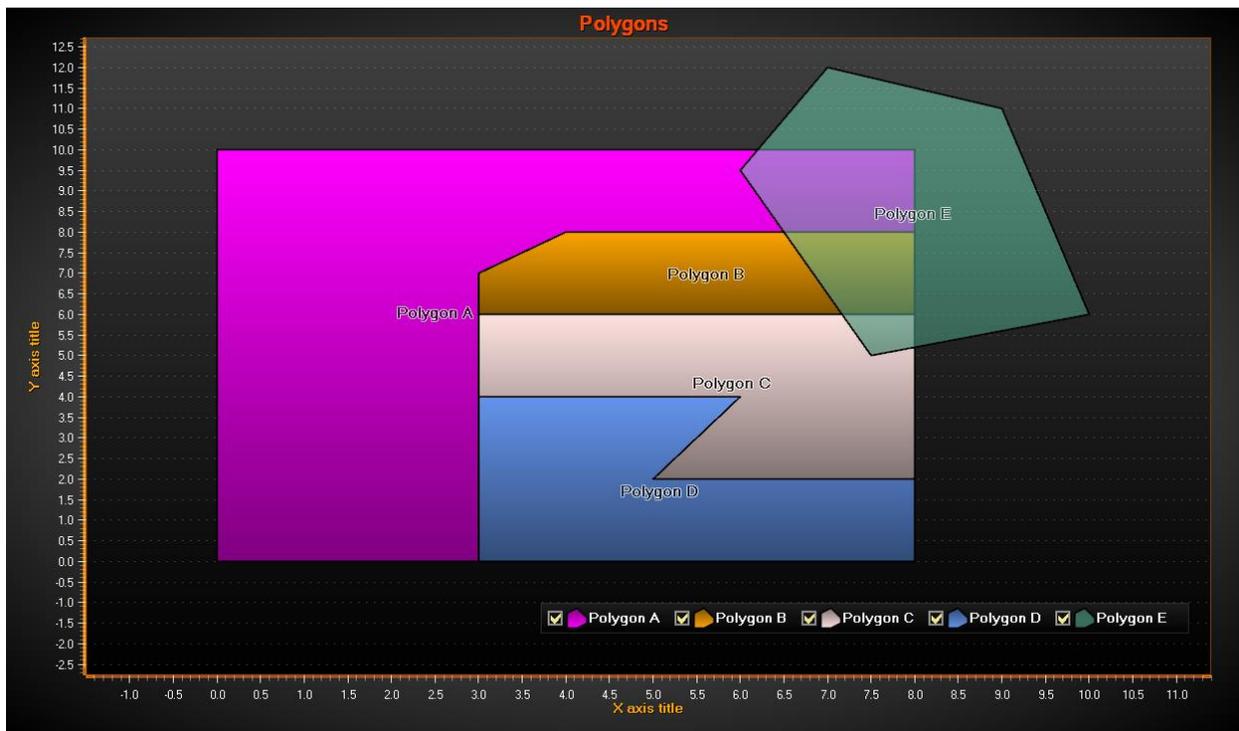


Figure 5-62. Several polygons.

5.14.1 Setting data to a Polygon

Set the path points in **Points** property. **PolygonSeries** has an automatic path closing feature. If the last point is not connected to the first point, the chart will do that automatically.

The following shows how to assign the points of the previous picture's transparent teal polygon's path:

```
polygon.Points = new PointDouble2D[] {  
    new PointDouble2D(7,12),  
    new PointDouble2D(6,9.5),  
    new PointDouble2D(7.5,5),  
    new PointDouble2D(10,6),  
    new PointDouble2D(9,11)};
```

5.14.2 Enabling complex / intersecting fills

Set ***IntersectionsAllowed = True*** to enable polygon path to intersect itself. Without this property enabled, when intersecting the path, the fill will appear all garbled up. By default, the property is ***False*** for performance reasons, as detecting and rendering intersection cases is heavy.

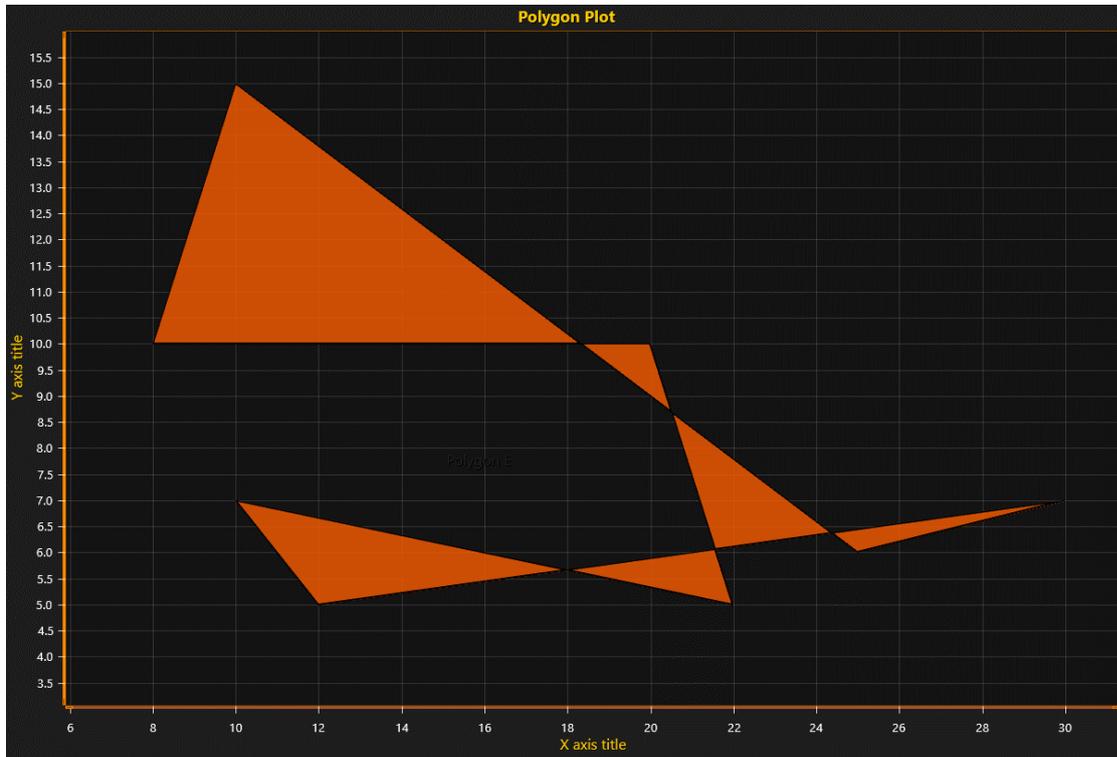


Figure 5-63. Polygon with intersecting path, with ***IntersectionsAllowed = True***.

5.15 LineCollections

A ***LineCollection*** is a collection of line segments. Each line segment is a line going from point A to B. One ***LineCollection*** can contain thousands of line segments. ***LineCollection*** is extremely efficient in rendering of thousands of distinct line segments, in contrast to ***PointLineSeries***, ***FreeformPointLineSeries*** or ***SampleDataSeries***. ***PointLineSeries***, ***FreeformPointLineSeries*** or ***SampleDataSeries*** are more efficient in rendering continuous polylines of millions of points.

Use ***LineStyle*** property to control the line color, style and width. Set the line segments in ***Lines*** property.

Add the ***LineCollection*** object in ***ViewXY.LineCollections*** list property.

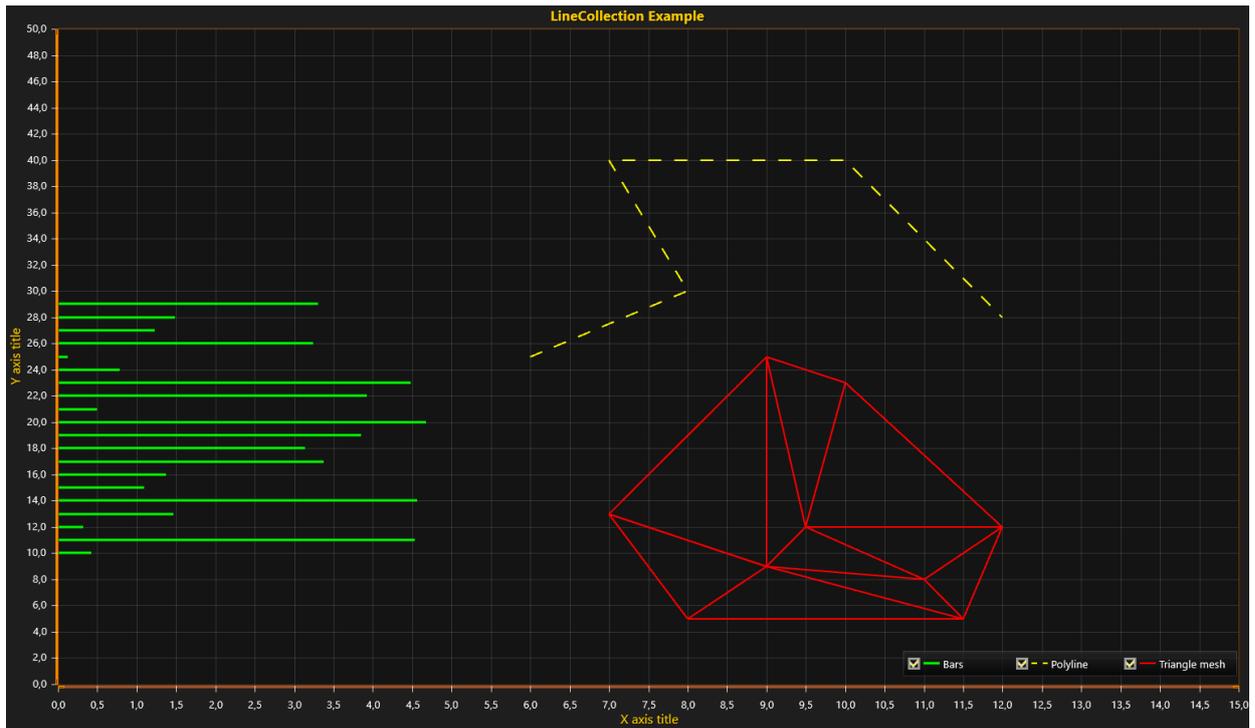


Figure 5-64. Three LineCollections in use. Green acts as very rapidly rendering bars, yellow as a polyline, and red as an arbitrary triangle wireframe mesh.

5.15.1 Setting data to a LineCollection

SegmentLine structure consists of four fields:

AX Start point, X
AY Start point, Y
BX End point, X
BY End point, Y

Add the **SegmentLines** array to **Lines** property as follows:

```
lineCollection.Lines = new SegmentLine[] {
    new SegmentLine(6,25,8,30),
    new SegmentLine(8,30,7,40),
    new SegmentLine(7,40,10,40),
    new SegmentLine(10,40,12,28) };
```

5.16 IntensityGridSeries

IntensityGridSeries allows visualizing M x N array of nodes, colored by assigned value-range palette. The colors between nodes are interpolated. *IntensityGridSeries* is an evenly-spaced, rectangular series in X and Y dimension. This series allows rendering contour lines, contour line labels, and wireframe as well.

Misc	
> AssignableXAxes	String[] Array
> AssignableYAxes	String[] Array
AssignXAxisIndex	0
AssignYAxisIndex	0
> ContourLineLabels	
> ContourLineStyle	
ContourLineType	ColorLine
> Data	IntensityPoint[,] Array
DisableDragToAnotherAxis	True
FastContourZoneRange	1
Fill	Paletted
FullInterpolation	False
IncludeInAutoFit	True
InitialValue	0
LegendBoxIndex	0
LegendBoxUnits	°C
LegendBoxValuesFormat	0
LegendBoxValueType	Number
LimitYToStackSegment	False
MouseHighlight	None
MouseInteraction	False
Optimization	DynamicData
PixelRendering	False
RangeMaxX	100
RangeMaxY	100
RangeMinX	0
RangeMinY	0
ShowInLegendBox	True
ShowNodes	False
SizeX	400
SizeY	240
> Stencil	
> Title	
ToneColor	■ Black
TraceMouseCell	True
> ValueRangePalette	
Visible	True
> WireframeLineStyle	
WireframeType	None

Figure 5-65. IntensityGridSeries properties.

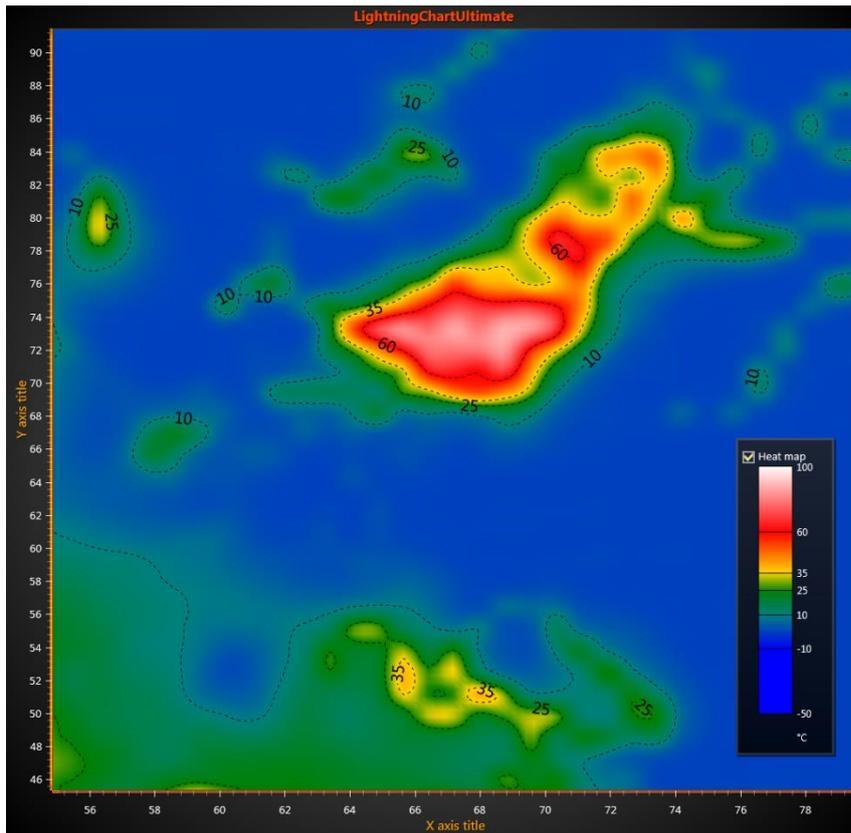


Figure 5-66. IntensityGrid series showing a heat map presentation. Legend box shows the value-range color palette.

The data is stored in **Data** property as two-dimensional array. Each array item is of type **IntensityPoint**. Store the data value of each node in **Value** field of **IntensityPoint** structure, which tells what color should be used from the **ValueRangePalette**.

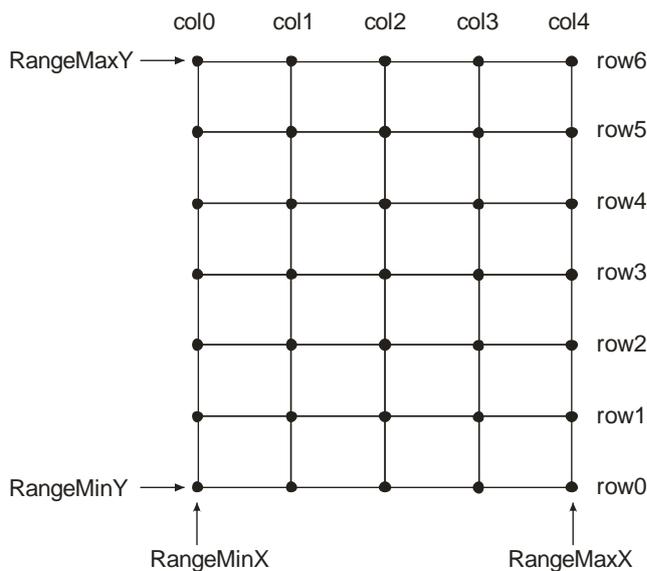


Figure 5-67. IntensityGridSeries nodes. SizeX = 5, SizeY = 7.

Node distances are automatically calculated as

$$\text{node distance } X = \frac{\text{RangeMaxX} - \text{RangeMinX}}{\text{SizeX} - 1}$$

$$\text{node distance } Y = \frac{\text{RangeMaxY} - \text{RangeMinY}}{\text{SizeY} - 1}$$

5.16.1 Setting intensity grid data

- Set X range by using **RangeMinX** and **RangeMaxX** properties, to order the minimum and maximum value of assigned X axis.
- Set Y range by using **RangeMinY** and **RangeMaxY** properties, to order the minimum and maximum value of assigned Y axis.
- Set **SizeX** and **SizeY** properties to give the grid a size as columns and rows.
- Set **Value** for each node:

Method, with Data array index

```
for (int nodeIndexX = 0; nodeIndexX < columnCount; nodeIndexX ++)  
{  
    for (int nodeIndexY = 0; nodeIndexY < rowCount; nodeIndexY ++)  
    {  
        intensityValue = //some height value.  
        gridSeries.Data[iNodeX, iNodeY].Value = intensityValue;  
    }  
}  
gridSeries.InvalidateData(); //Notify new values are ready and to refresh
```

Alternative method, usage of SetDataValue

```
for (int nodeIndexX = 0; nodeIndexX < columnCount; nodeIndexX ++)  
{  
    for (int nodeIndexY = 0; nodeIndexY < rowCount; nodeIndexY ++)  
    {  
        intensityValue = //some height value  
        gridSeries.SetDataValue(nodeIndexX, nodeIndexY,  
            0, //X value is irrelevant in grid  
            0, //Y value is irrelevant in grid  
            intensityValue,  
            Color.Green); //Source point colors are not used in this  
            example, so use any color here  
    }  
}  
gridSeries.InvalidateData(); //Notify new values are ready and to refresh
```

Setting Values only to existing grid

When the geometry of IntensityMesh, or SizeX or SizeY for IntensityGrid series doesn't change while data is changing rapidly, it is most advantageous to use **SetValuesData** method. Since it accepts Double[][] format data array, scrolling or re-ordering rows or columns is quick. Especially when combined with **PixelRendering** property (see 5.16.4), it is a very effective approach for high-resolution scrolling spectrogram visualization. Note that when **PixelRendering** is **disabled** with external data array set by SetValuesData, **Data** property can't be null.

Setting Colors only to existing grid

When the geometry of IntensityMesh, or SizeX or SizeY for IntensityGrid series doesn't change while data is changing rapidly, it is most advantageous to use **SetColorsData** method. It accepts int[][] format values, i.e. ARGB values that GPU accepts directly. With this kind of data array, scrolling or re-ordering rows or columns is quick. Especially when combined with **PixelRendering** property (see 5.16.4), it is a very effective approach for high-resolution scrolling spectrogram visualization. Note that when **PixelRendering** is **disabled** with external data array set by SetColorsData, **Data** property can't be null.

5.16.2 Creating intensity grid data from bitmap file

Create a surface from a bitmap image. Use **SetHeightDataFromBitmap** method to achieve this. The series **Data** array property gets the size of the bitmap size (if no anti-aliasing or resampling is used). For each bitmap image pixel, Red, Green and Blue values are summed. The greater the sum, the greater will be the data value for that node. Black and dark colors get lower values and bright and white colors get higher values.

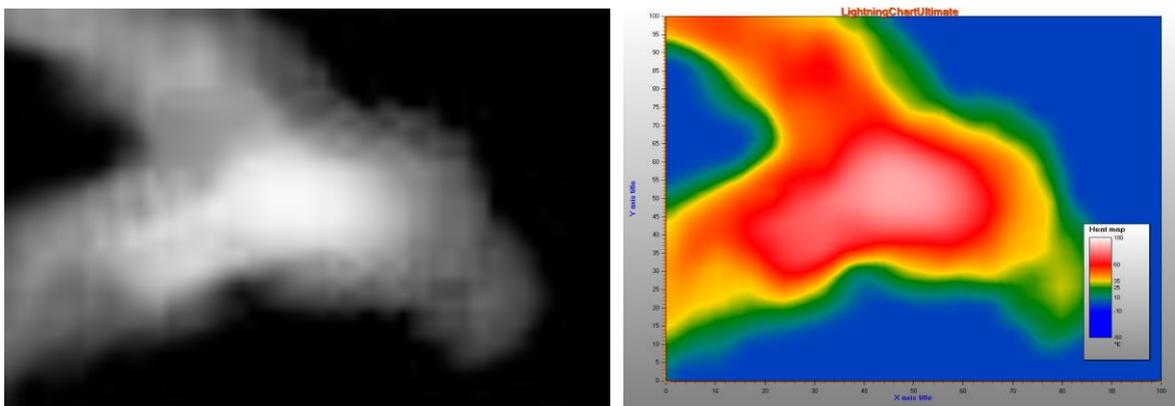


Figure 5-68. Source bitmap and calculated intensity values data. Dark values stay low and bright values get higher values.

5.16.3 Fill styles

Use **Fill** property to select the filling style. The following options are available

- **None**: By using this, no filling is applied. This is the selection to use with wireframe mesh or plain contour lines.
- **FromSurfacePoints**: The colors of the Data property nodes are used.
- **Toned**: ToneColor applies
- **Paletted**: See chapter 5.16.5.

Enable **FullInterpolation** property to use enhanced interpolation method in the fill. Note that it will cause more CPU and GPU usage. By using full interpolation, the fill quality is better, but can be seen only when the data array size is quite small.

5.16.4 Rendering as pixel map

By enabling **PixelRendering** property, the nodes are rendered as pixels, or rectangles. This is a very high-performance rendering style e.g. for real-time high-resolution thermal imaging applications. Note that when this rendering mode is selected, many other options are disabled, such as contour lines, wireframe and interpolation. If logarithmic axes are used, the logarithmic transformation is only applied to corners of the series, the pixels in the bitmap remain evenly spaced and no logarithmic transformation is applied to them.

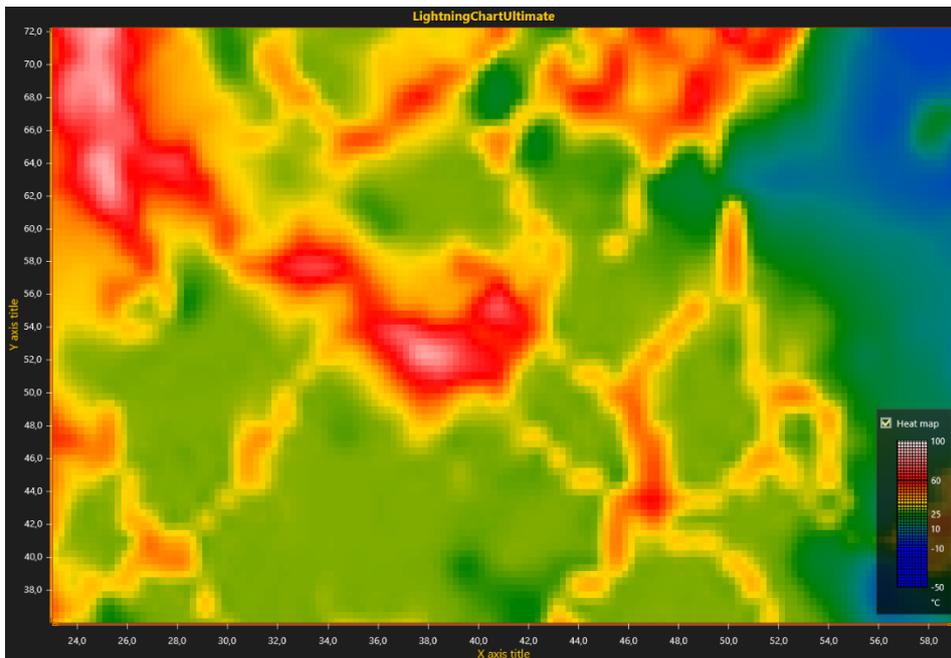


Figure 5-69. PixelRendering = true.

5.16.5 ValueRangePalette

With **ValueRangePalette** property, define color steps for value coloring. **ValueRangePalette** can be used for:

- **Fill** (see chapter 5.16.3)
- **Wireframe** (see chapter 5.16.6)
- **Contour lines** (see chapter 5.16.7)

Define several steps for contour palette. Each step has a height value and the corresponding color.

Note! 20 steps are precompiled and loaded fast. With higher step counts, several seconds delay can be expected when initializing the chart.

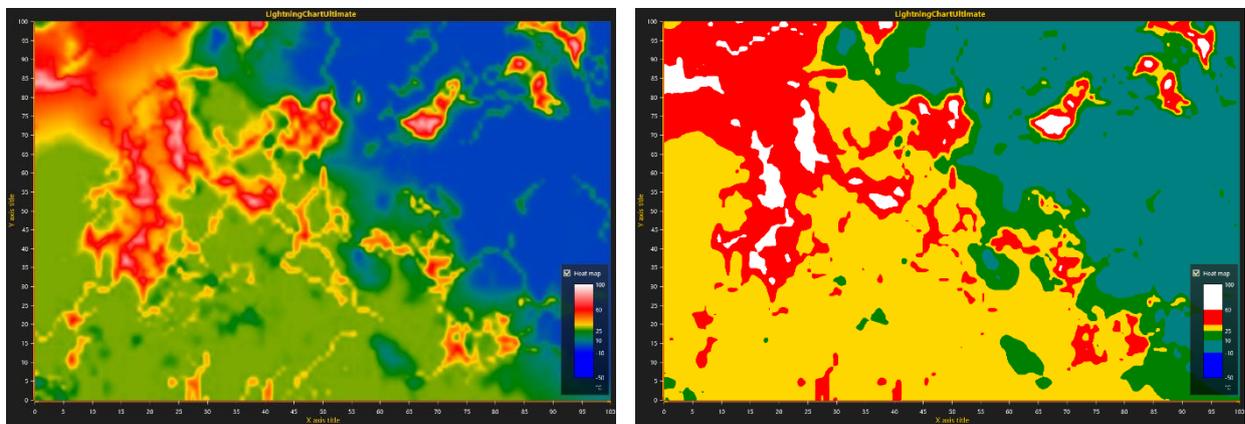


Figure 5-70. On the left, **IntensityGridSeries Fill** is set to **Paletted** and **Palette Type** is set to **Gradient**. On the right, **Palette Type** is set to **Uniform**.

The palette is defined with **MinValue**, **Type** and **Steps** properties. For **Type**, there are two choices: **Uniform** and **Gradient**. The contour palettes (check the legend boxes) of the previous figures show:

- **MinValue:** -50
- **Type:** Uniform
- **Steps:**
 - Steps[0]: MaxValue: -10, Color: Blue
 - Steps[1]: MaxValue: 10, Color: Teal
 - Steps[2]: MaxValue: 25, Color: Green
 - Steps[3]: MaxValue: 35, Color: Yellow
 - Steps[4]: MaxValue: 60, Color: Red
 - Steps[5]: MaxValue: 100, Color: White

The values below the first step value are colored with the first step's color.

5.16.6 Wireframe

Use **WireframeType** to select the wireframe style. The options are:

- **None**: no wireframe
- **Wireframe**: a solid color wireframe. Use **WireframeLineStyle.Color** to set the color
- **WireframePaletted**: the wireframe coloring follows **ValueRangePalette** (see chapter 5.16.5)
- **WireframeSourcePointColored**: the wireframe coloring follows the color of grid nodes
- **Dots**: solid color dots are drawn in the grid node positions
- **DotsPaletted**: dots are drawn in the grid node positions and colored by **ValueRangePalette**
- **DotsSourcePointColored**: dots are drawn in the grid node positions, coloring follows the color of grid nodes

The wireframe line style (color, width, pattern) can be edited by using **WireframeLineStyle**.

Note! Palette colored wireframe lines and dots are available only when **WireframeLineStyle.Width = 1** and **WireframeLineStyle.Pattern = Solid**.

5.16.7 Contour lines

Contour lines can be used with fill and wireframe properties. By setting **ContourLineType** property, contour lines can be drawn with different styles:

- **None**: no contour lines are shown
- **FastColorZones**: The lines are drawn as thin zones on palette step end. Allows very powerful rendering, which suits very well for continuously updated or animated surface. Steep value changes are shown as thin line, while gently sloping height differences are shown with thick zone. Each line uses the same color defined with **ContourLineStyle.Color** property. The zone width can be set by **FastContourZoneRange** property. The value is in Y axis range.
- **FastPalettedZones**: Like **FastColorZones**, but line coloring follows **ValueRangePalette** options (see chapter 5.16.5).
- **ColorLine**: Like **FastColorZones**, but the contour lines are actual lines. Rendering takes longer and is not recommended for continuously updated or animated surface. The line width can be adjusted with **ContourLineStyle.Width** property.
- **PalettedLine**: Like **ColorLine**, but line coloring follows **ValueRangePalette** options.

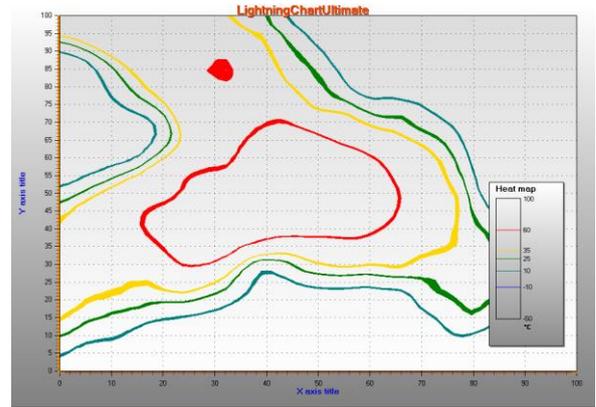
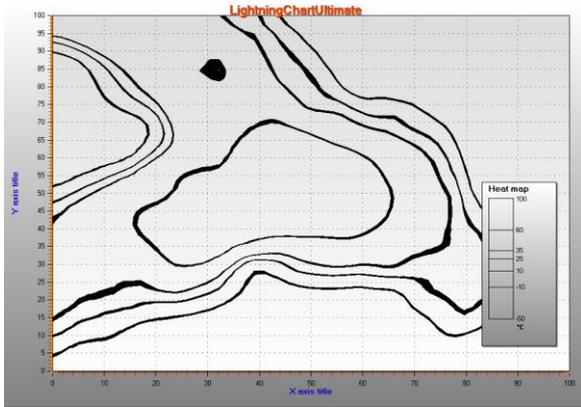


Figure 5-71. On the left, ContourLineType = FastColorZones. On the right, ContourLineType = FastPalettedZones.

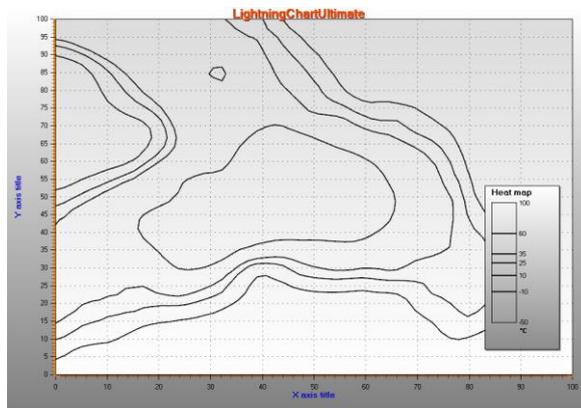


Figure 5-71. On the left, ContourLineType = ColorLine. On the right, ContourLineType = PalettedLine

5.16.8 Contour line labels

When contour lines are visible, numeric values can be shown within the line paths.

ContourLineLabels	
Color	Black
Font	Segoe UI, 15pt, style=Bold
LabelsNumberFormat	0.0
Visible	True

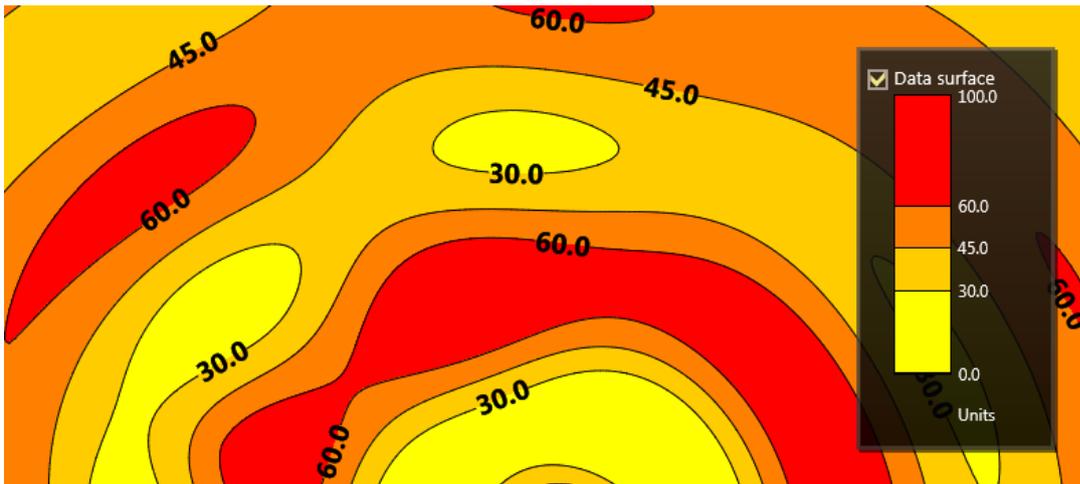


Figure 5-72. The properties of ContourLineLabels and the result

Use *LabelsNumberFormat* for custom string formatting, for example setting the number of decimals.

5.17 IntensityMeshSeries

IntensityMeshSeries is almost similar to *IntensityGridSeries*. The biggest difference is that series nodes can be positioned arbitrarily in X-Y space. In other words, the series does not have to be rectangular. Wireframe lines can be set visible with *WireframeType* property, and nodes can be shown by setting *ShowNodes* true.

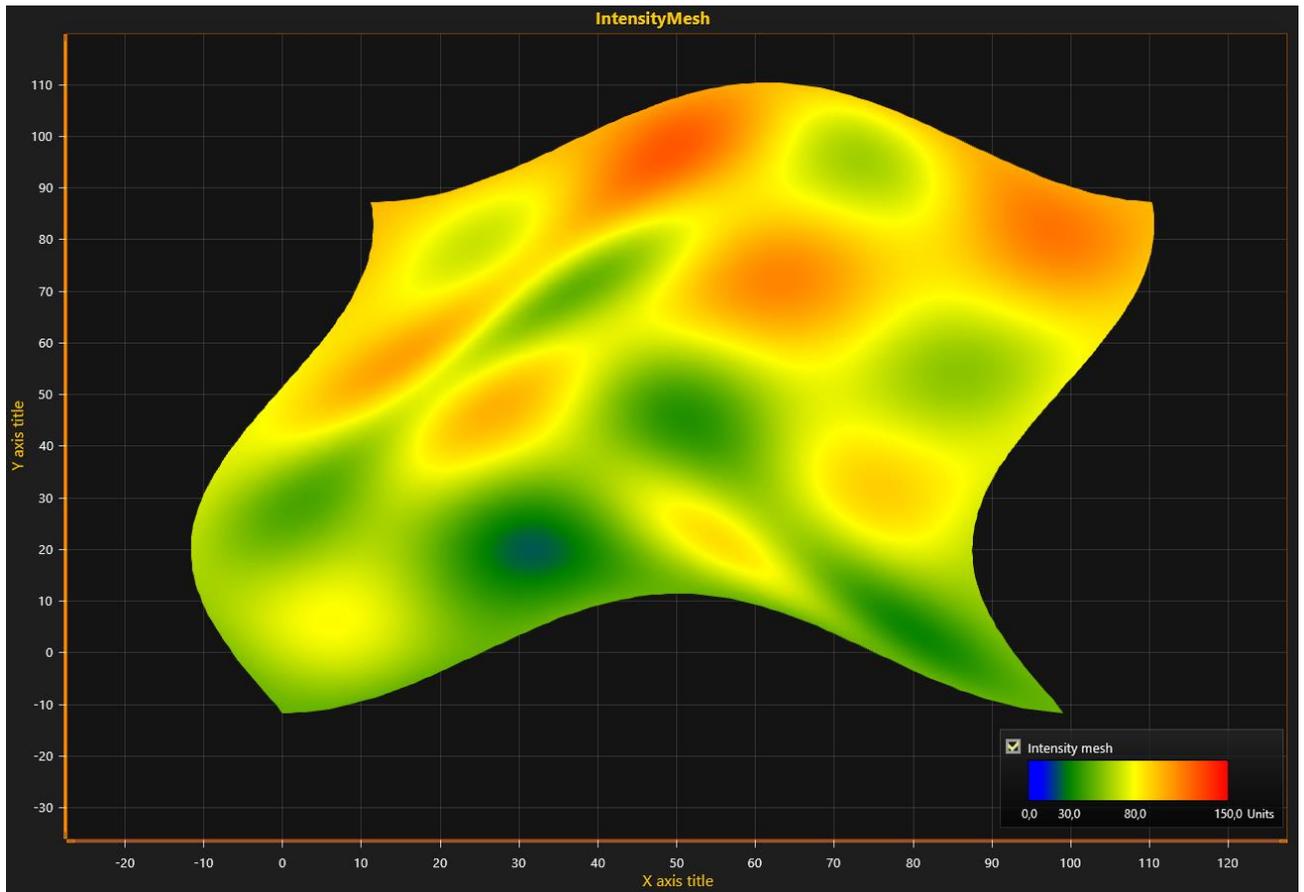


Figure 5-73. IntensityMeshSeries with freely positioned X and Y values for each node. WireframeType = Wireframe and ShowNodes = true.

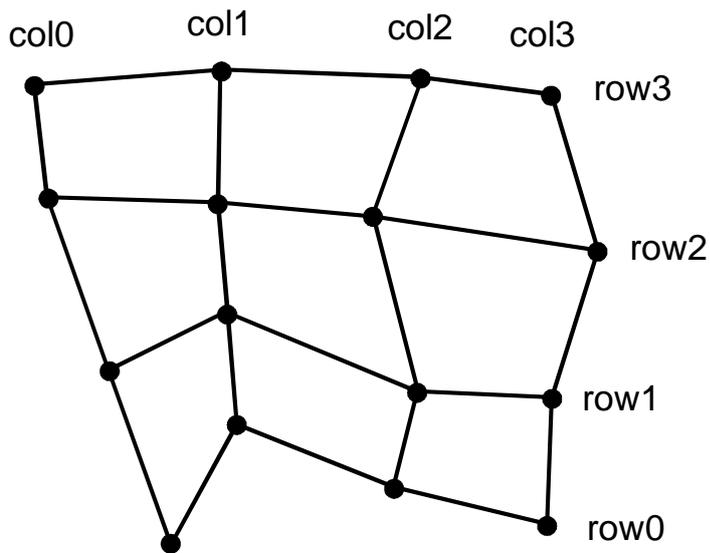


Figure 5-74. Intensity mesh nodes. SizeX = 4, SizeY =4.

5.17.1 Setting intensity mesh data, when geometry changes

Follow these instructions, when the **X**, **Y** and **Value** fields are updated in the same time.

- Set **SizeX** and **SizeY** properties to give the mesh a size as columns and rows.
- Set X, Y and Y values for all nodes:

Method, with Data array index

```
for (int nodeIndexX = 0; nodeIndexX < columnCount; nodeIndexX ++)  
{  
    for (int nodeIndexY = 0; nodeIndexY < rowCount; nodeIndexY ++)  
    {  
        meshSeries.Data[nodeIndexX, nodeIndexY].X = xValue;  
        meshSeries.Data[nodeIndexX, nodeIndexY].Y = yValue;  
        meshSeries.Data[nodeIndexX, nodeIndexY].Value = value;  
    }  
}  
meshSeries.InvalidateData(); //Notify new values are ready to refresh
```

Alternative method, usage of SetDataValue

```
for (int nodeIndexX = 0; nodeIndexX < columnCount; nodeIndexX ++)  
{  
    for (int nodeIndexY = 0; nodeIndexY < rowCount; nodeIndexY ++)  
    {  
        meshSeries.SetDataValue(nodeIndexX, nodeIndexY,  
            xValue,  
            yValue,  
            value,  
            Color.Green); //Source point colors are not used in this  
                           example, so use any color here  
    }  
}  
meshSeries.InvalidateData(); //Notify new values are ready to refresh
```

5.17.2 Setting intensity mesh data, when geometry does not change

Follow these instructions, when only the **Value** fields of **Data** array **IntensityPoint** structures are updated. This is the performance optimized way for updating data for example in thermal imaging or environmental data monitoring solutions, where **X** and **Y** values of each node stay at the same location.

5.17.2.1 Creating the series and its geometry

- Set **Optimization** to **DynamicValuesData**
- Set **SizeX** and **SizeY** properties to give the mesh a size as columns and rows.
- Set X, Y and Y values for all nodes:

```
for (int nodeIndexX = 0; nodeIndexX < columnCount; nodeIndexX ++)  
{  
    for (int nodeIndexY = 0; nodeIndexY < rowCount; nodeIndexY ++)  
    {  
        meshSeries.Data[nodeIndexX, nodeIndexY].X = xValue;  
        meshSeries.Data[nodeIndexX, nodeIndexY].Y = yValue;  
        meshSeries.Data[nodeIndexX, nodeIndexY].Value = value;  
    }  
}  
meshSeries.InvalidateData(); //Rebuild geometry from nodes and repaint
```

5.17.2.2 Updating the values periodically

- Set only values for all nodes:

```
for (int nodeIndexX = 0; nodeIndexX < columnCount; nodeIndexX ++)  
{  
    for (int nodeIndexY = 0; nodeIndexY < rowCount; nodeIndexY ++)  
    {  
        meshSeries.Data[nodeIndexX, nodeIndexY].Value = value;  
    }  
}  
meshSeries.InvalidateValuesDataOnly(); //Only data values are updated
```

5.18 Bands

Bands can be considered as series. They have the same user interface actions as other series, but one band series contains only one band. A band is a vertical or horizontal area reaching from a margin across to another. A band can be bound to a Y axis or X axis using the **Binding** property. If the band is bound to Y axis, **AssignYAxisIndex** property must also be set. If the series is bound to X axis, ignore **AssignYAxisIndex** property, or set it as unassigned (-1).

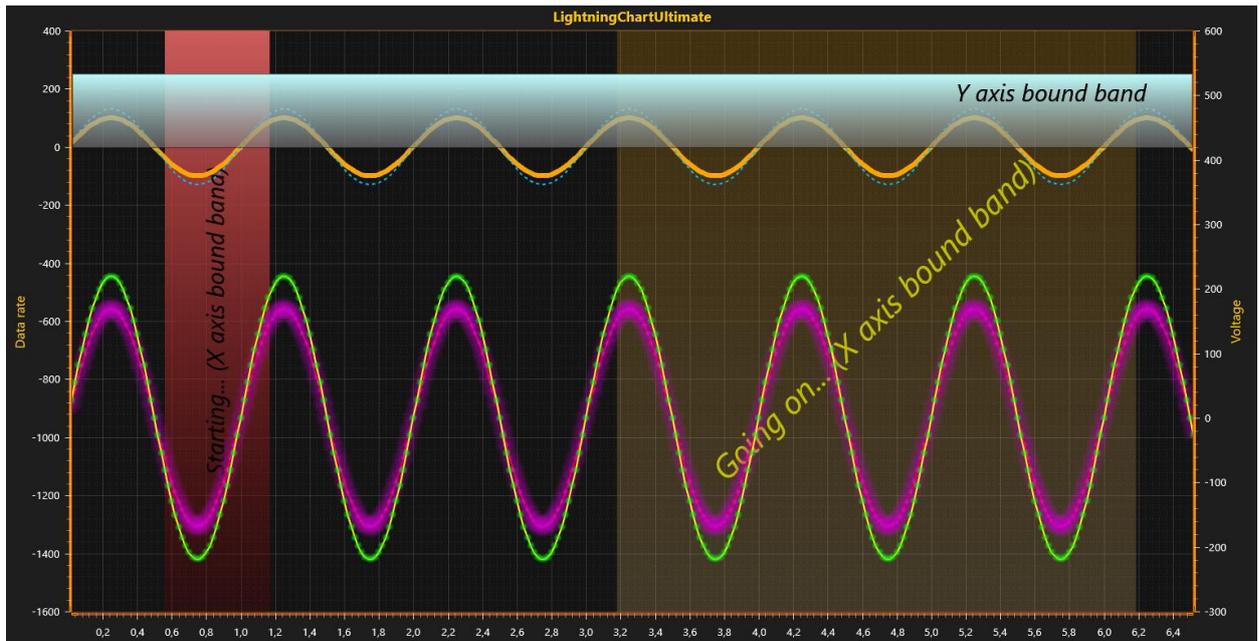


Figure 5-75. A couple of bands with line series

If the band should be behind the line or bar series, set **Behind** property true. Band edges are set by **ValueBegin** and **ValueEnd** properties, which are values of the bound axis. Band can be dragged to another location with mouse. Resize the band by dragging it from the edge, which updates then the dragged edge value, **ValueBegin** or **ValueEnd**.

5.19 Constant lines

Like bands, constant lines can be considered as series. Constant lines are bound to Y axis, and it represents one horizontal line, ranging from graph left edge to right edge. Set the level in **Value** property. Constant lines can be vertically moved by dragging with mouse. By setting **Behind** property true, the constant line is drawn behind line and bar series, otherwise it is drawn in front of them.

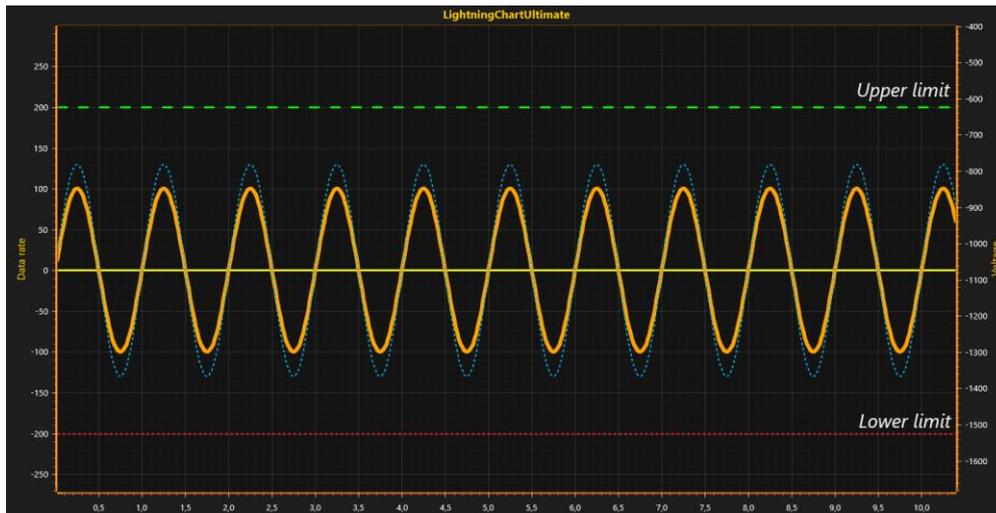


Figure 5-76. Some constant lines around a sine line series.

5.20 Annotations

Annotations allows displaying mouse-interactive text labels or graphics anywhere in the chart area. Annotations can be moved around by mouse, resized, rotated, their target and location can be changed etc. Alternatively, they can be controlled by code. Annotations are great also when custom graphics must be rendered on the screen, as they can be rendered in different styles and shapes. Create **AnnotationXY** objects in **ViewXY.Annotations** collection.

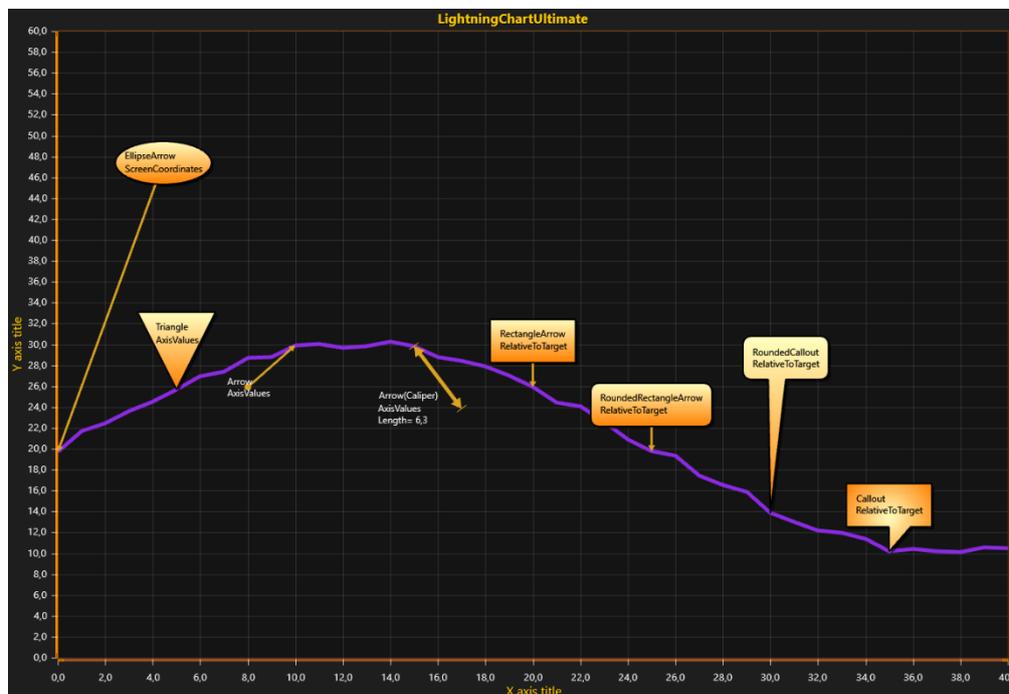


Figure 5-77. AnnotationXY objects with various styles, placed around a line series. Use **Style** property to select the shape.

By moving mouse over an annotation, it goes into mouse-interactive edit state, allowing relocating the annotation, resizing it, rotating it, and determining where the arrow points to.



Figure 5-78. Move mouse over the annotation to enter the editing state. Move mouse away to leave the edit state.

5.20.1 Controlling target and location

Target is the ending point of the arrow, the point that the arrow or callout tip points to. **Target** can be set in axis values or in screen coordinates. Use **TargetCoordinateSystem** to select between **AxisValues** or **ScreenCoordinates**. When **AxisValues** is selected, **TargetAxisValues** property sets where the arrow line points to (end of the arrow line). Use **TargetScreenCoords** to set it in screen coordinates instead.

Location is the starting point of the arrow. It can be set by screen coordinates, axis values, or as relative offset from **Target**. Use **LocationCoordinateSystem** to select, and **LocationScreenCoords**, **LocationAxisValues** or **LocationRelativeOffset** to control the location by the selected method. **Location** is also the center point of text area rotation.

Anchor property controls how the text area is placed at **Location**. By setting **Anchor.X** = 0.5 and **Anchor.Y** = 0.5, the beginning of the arrow is in the middle. When setting **Anchor.X** 0.1 and **Anchor.Y** = 0.25, arrow start is near the upper left corner as the following figure illustrates:

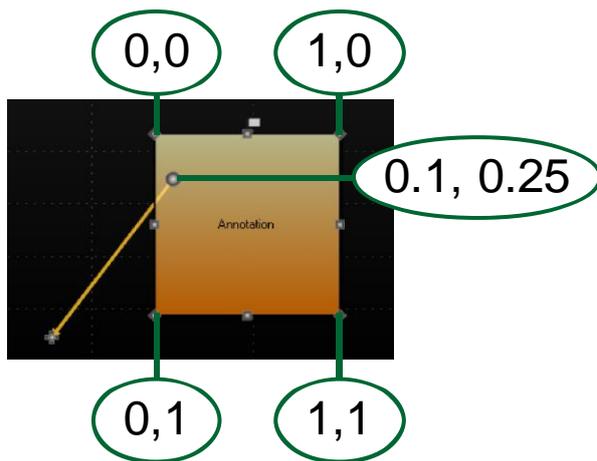


Figure 5-79. Anchor values explained. Current Anchor.X = 0.1 and Anchor.Y = 0.25. When the anchor values are between 0...1, the arrow start point is inside the text area.

5.20.2 Using mouse to move, rotate and resize

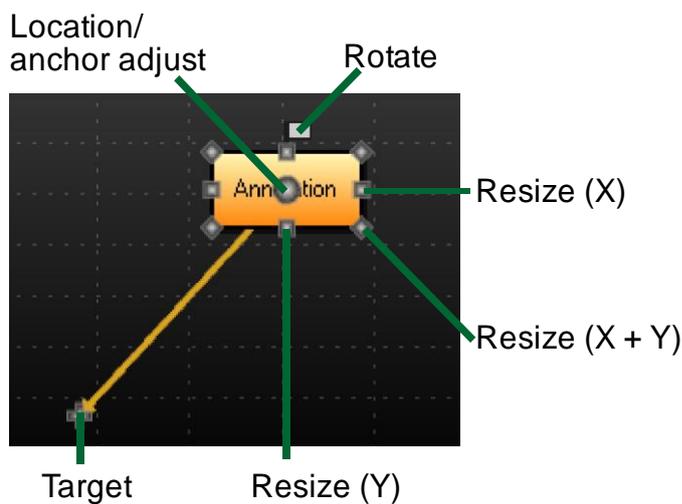


Figure 5-80. Annotation mouse-interactive nodes.

Drag from **Target** to move the end of the arrow. Drag from text area to set new **Location**. By dragging from round location/anchor node, **Anchor** and **Location** properties can be adjusted at the same time, keeping the text box in the same place.

By holding **Shift** key down while dragging from X or Y resize node, a symmetrical operation is used, both sides are adjusted at same time. By holding **Shift** key down while dragging from a corner resize node (X+Y), resizing maintains the aspect ratio. In rotate operation, **Shift** key snaps the rotate angle to nearest multiple of 15 degrees.

5.20.3 Adjusting appearance

Select the annotation shape by setting **Style** property. The options are: **Rectangle**, **RectangleArrow**, **RoundedRectangle**, **RoundedRectangleArrow**, **Arrow**, **Callout**, **RoundedCallout**, **Ellipse**, **EllipseArrow**, **Triangle** and **TriangleArrow**.

With styles with arrow, use **ArrowLineStyle**, **ArrowStyleBegin** and **ArrowStyleEnd** to control the arrow design. As arrow end styles, there are options: **None**, **Square**, **Arrow**, **Circle** and **Caliper**.

Use **Fill** to modify the fill of the annotation. The appearance of the editing state mouse-interactive nodes can be changed from **NibStyle**. **TextStyle** controls the font settings and text alignment inside the text area. **BorderLineStyle** and **CornerRoundRadius** control the border line appearance.

5.20.4 Size settings

Sizing property controls how the annotation text box is to be sized:

- **Automatic** adjusts the size by the contents, and leaves **AutoSizePadding** space to the borders.
- **AxisValuesBoundaries** allows the size of the annotation to be set by axis values. Use **AxisValuesBoundaries.XMin**, **XMax**, **YMin** and **YMax** for defining them.
- **ScreenCoordinates** enables settingsize by the screen coordinates. Use **SizeScreenCoords.Height** and **Width**.

5.20.5 Keeping text area visible

When **KeepVisible** is enabled, the annotation text area is forced inside the graph. The annotation won't move outside the graph when moving it by mouse or code. When panning the graph view or adjusting axes, the annotations are repositioned to show inside the graph.

5.20.6 Displaying annotation over axes

By setting **RenderBehindAxes = True**, annotation is shown under axes. All clipping and Z ordering features are not feasible in that case. **RenderBehindAxis** has no effect if **ClipInsideGraph** is set true.

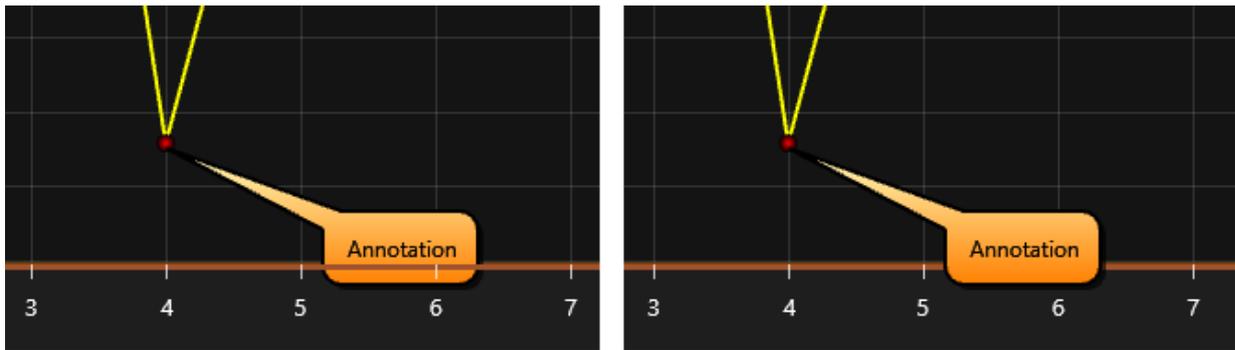


Figure 5-81. On the left `RenderBehindAxes = True`, on the right `RenderBehindAxes = False`. `ClipInsideGraph` is set `False` in both cases.

5.20.7 Clipping inside graph

When ***ClipInsideGraph*** is enabled, the annotation is clipped inside the graph. When it's disabled, the annotation is rendered also in the margin area of the chart.

By enabling ***ClipWhenSweeping***, the annotation doesn't show up in the sweeping gap area when ***ScrollMode*** is set ***Sweeping***.

5.20.8 Controlling the Z order

By setting ***Behind*** property to its default value, ***False***, the annotation appears on top of series. By setting it ***True***, it is rendered before the series, thus appearing under them.

The annotations appear in the order they exist in Annotations list, while keeping the ***Behind*** filter as a master controller. Annotations Z order can be changed quickly by using ***ChangeOrder*** method of annotation for example in a mouse event handler. The options for order change are:

- ***BringToFront*** brings the annotation to topmost
- ***SendToBack*** sends to back
- ***MoveBack*** moves one step backwards
- ***MoveFront*** moves one step forwards

5.20.9 LayerGrouping performance optimization

When having hundreds of annotations with visible text, the delay of text rendering starts to play a significant role. By default, text rendering follows the Z order, keeping the text firmly within an annotation.

The performance can be improved by setting **LayerGrouping = True**, and the chart will use only two flat annotation text layers. One for annotations with **Behind** set to **True**, and other for annotations with **Behind** set to **False**. It greatly improves performance. On the other hand, the text will be rendered wrong if there are other annotations overlapping others.

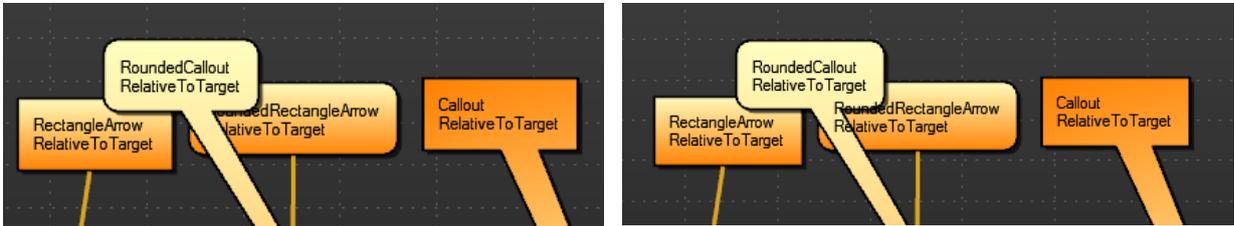


Figure 5-82 On the left **LayerGrouping = False**. On the right, **LayerGrouping = True**, Z order of texts is lost.

When using **Style = Arrow** or by setting the annotation fill not visible, the restriction of Z order typically doesn't show up.

5.20.10 Converting between axis values and screen coordinates

In some cases, **Location** or **Target** may be wanted to be defined in mixed configuration. X in screen coordinates and Y in axis values, or vice versa. Axes have **ValueToCoord** method for converting an axis value to a screen coordinate, and **CoordToValue** to convert a screen coordinate to an axis value as described in chapter 5.2.11.

5.21 Legend box

Starting from v.8, ViewXY supports multiple legend boxes in the same graph. Insert these legend boxes in **ViewXY.LegendBoxes** collection.

▼ Misc	
AlignmentInSegmentGap	Near
AlignmentInVerticalMargin	Center
AllowMouseResize	True
AutoSize	True
BorderColor	<input type="color" value="#402552"/> 40. 255. 255. 255
BorderWidth	1
Categorization	None
CategoryColor	<input type="color" value="white"/> White
> CategoryFont	Segoe UI, 10pt, style=Bold
CheckBoxColor	<input type="color" value="#140255"/> 140. 255. 255. 255
CheckBoxSize	15
CheckMarkColor	<input type="color" value="khaki"/> Khaki
> Fill	
Height	31
HighlightSeriesOnTitle	True
HighlightSeries TitleColor	<input type="color" value="yellow"/> Yellow
> IntensityScales	
Layout	Horizontal
MouseHighlight	Simple
MouseInteraction	True
MoveByMouse	True
MoveFromSeries Title	True
> Offset	
Position	Segment BottomRight
ScrollBarVisibility	Both
SegmentIndex	0
Series TitleColor	<input type="color" value="white"/> White
> Series TitleFont	Segoe UI, 10pt
> Shadow	
ShowCheckboxes	True
ShowIcons	True
UnitsColor	<input type="color" value="white"/> White
> UnitsFont	Segoe UI, 9pt
UseSeries TitlesColors	False
ValueLabelColor	<input type="color" value="white"/> White
> ValueLabelFont	Segoe UI, 9pt
Visible	True
Width	303

Figure 5-83. Extensive LegendBoxXY property tree.

5.21.1 Hiding / showing a series from legend box

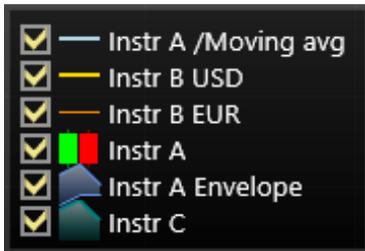


Figure 5-84. Legend box shows series titles and icons. Hide a series by deselecting the series checkbox.

5.21.2 Preventing a series from listing itself in the legend box

If a specific series should not be listed in the legend box, set *series.ShowInLegendBox* = **False**, for that series.

5.21.3 Selecting in which legend box to show a specific series

Use *series.LegendBoxIndex* to select the preferred legend box. Series can appear only in one legend box. Default index is 0 for all series, meaning they will all appear in the same legend box unless stated otherwise.

5.21.4 Selecting in which graph segment to show a legend box

Use *SegmentIndex* to control in which segment to show the legend box. It applies only to segment-based **Position** options.

5.21.5 Hiding check boxes

To hide the check boxes, set *ShowCheckBoxes* = **False**.

5.21.6 Hiding icons

To hide the icons, set *ShowIcons* = **False**.

5.21.7 Hiding intensity series palette scales

To hide the palette scale, set **IntensityScales.Visible = False**. To resize it, set **ScaleSizeDim1** and **ScaleSizeDim2** properties.



Figure 5-85. LegendBox.IntensityScales.Visible = 'False' in the bottom picture.

5.21.8 Controlling positions

Legend boxes can be placed automatically or manually. Automatic placement allows them to be aligned to the left/top/right/bottom side of the graph segments, or on margins. Control the position with **Position** property. Position options are: **TopCenter**, **TopLeft**, **TopRight**, **LeftCenter**, **RightCenter**, **BottomLeft**, **BottomCenter**, **BottomRight**, **Manual**.

If the view is divided to several segments, legend boxes can be aligned based to the segment it belongs to (use **SegmentIndex** to control this). For segment-based controlling there are the following options: **SegmentTopLeft**, **SegmentTopCenter**, **SegmentTopRight**, **SegmentBottomLeft**, **SegmentBottomCenter**, **SegmentBottomRight**, **SegmentLeftMarginCenter**, **SegmentRightMarginCenter**.

Offset property shifts the position by given amount *from the position determined by Position* property.

```
// Setting legend box position, offset shifts from RightCenter position
chart.ViewXY.LegendBoxes[0].Position = LegendBoxPositionXY.RightCenter;
chart.ViewXY.LegendBoxes[0].Offset = new PointIntXY(-15, -70);
```

Manual positioning calculates the offset from the top-left corner of the legend box to the view's top-left corner. Note that this differs from **TopLeft** option, which is calculated from the top of the graph area.

Note that when moving or resizing legend box, its **Position** is set to **Manual**, and **Offset** property is updated to reflect the new position.

Automatic legend box alignment is disabled until setting **Position** back to an option other than '**Manual**'. Since **Offset** is not updated when switching between **Position** options, legend box may seem to disappear sometimes (it is located outside the view). Fix this by setting **Offset** back to 0, 0.

5.21.9 Allocating space for legend boxes between graph segments

When setting `ViewXY.AutoSpaceLegendBoxes = True`, additional space between segments will be allocated to fit the legend boxes in them.

Note that also `ViewXY.AxisLayout.SegmentsGap` is allocated between segments.

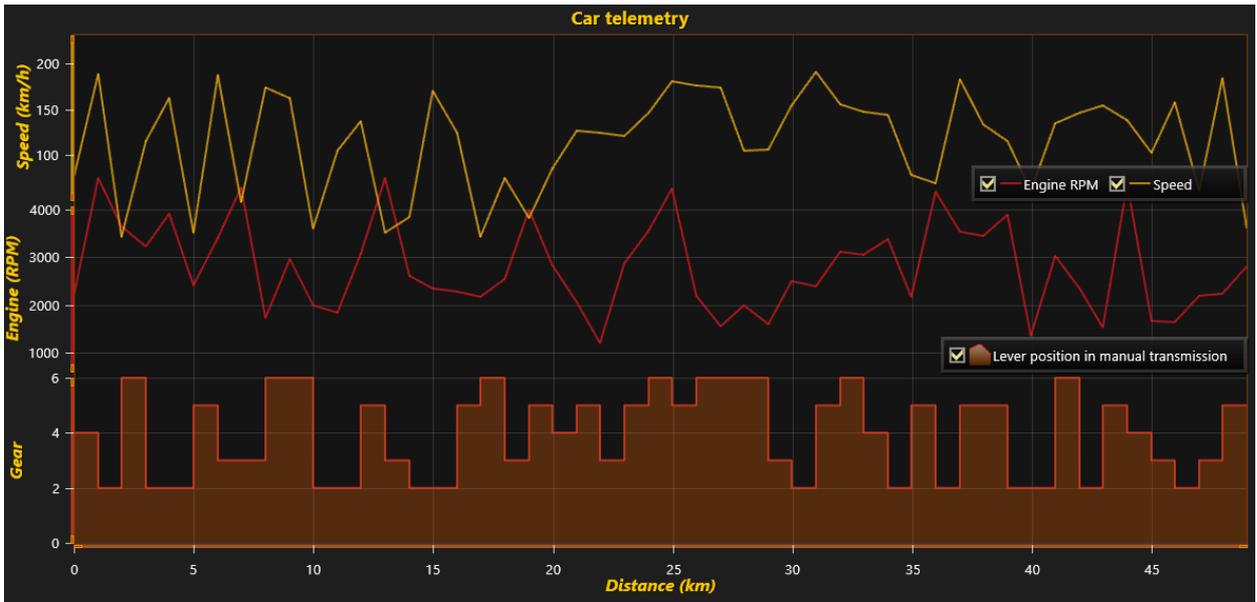


Figure 5-86. Position = SegmentBottomRight. AutoSpaceLegendBoxes = False.

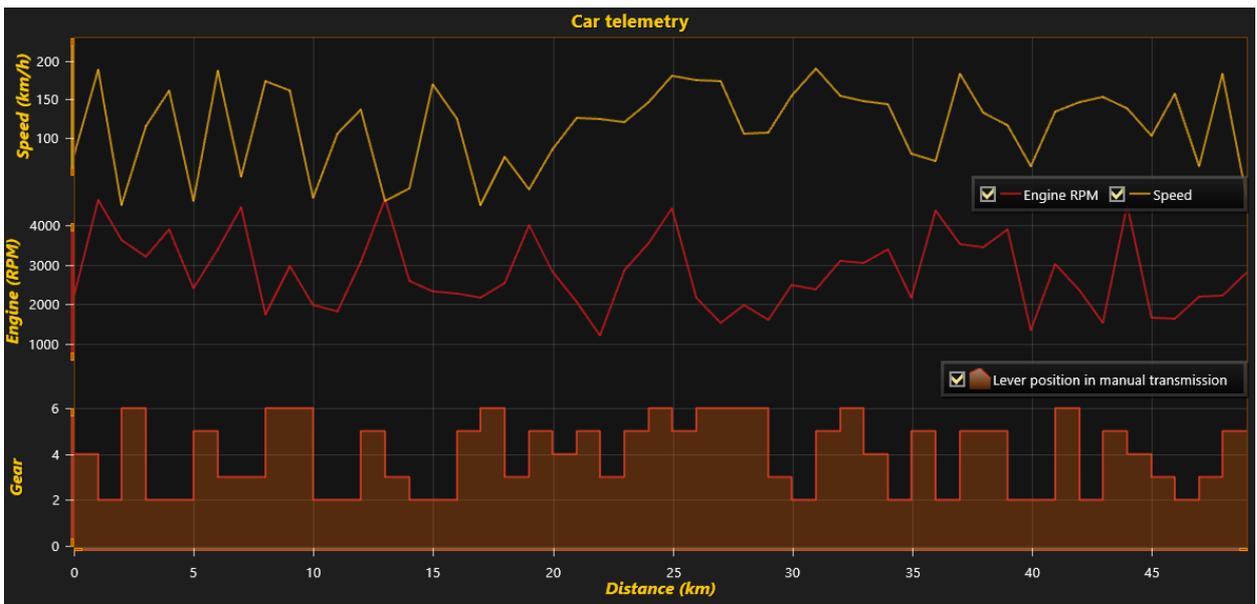


Figure 5-87. Position = SegmentBottomRight. AutoSpaceLegendBoxes = True.

5.21.10 Alignment of legend boxes in segment gap

To align legend box vertically near the specified segment, set *AlignmentInSegmentGap = Near*. To align it vertically to center of the gap between segments, set *AlignmentInSegmentGap = Center*.

5.21.11 Horizontal alignment of several legend boxes sharing the same margin

AlignmentInVerticalMargin property has *Left/Center/Right* options. The property controls horizontal positioning of legend boxes set to the same vertical margin.

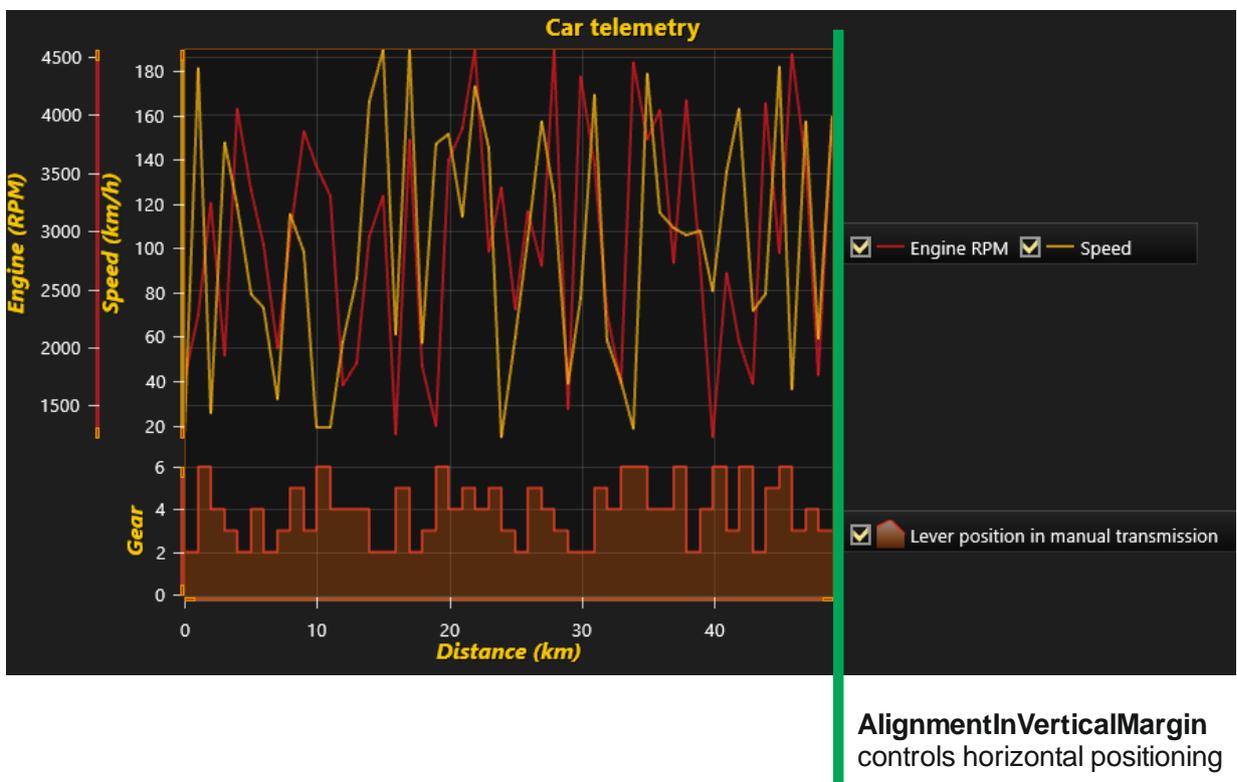


Figure 5-88. *AlignmentInVerticalMargin = Left* set for both Legend boxes.

5.21.12 Resizing and moving legend boxes

The legend boxes support resizing and scroll bars. Grab from the edge to resize it.

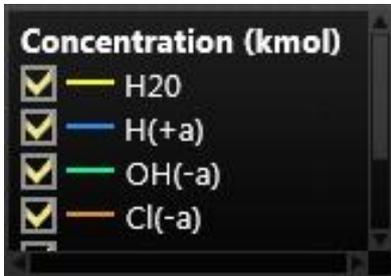


Figure 5-90. Scrollbars in a legend box.

Note that when moving or resizing legend box, its **Position** is set to **Manual**, and **Offset** property is updated to reflect the new position (see chapter 5.21.8).

5.22 Zooming and panning

Use **ZoomPanOptions** to control the zooming and panning settings.

ZoomPanOptions	
AltEnabled	True
AspectRatioOptions	
AspectRatio	Off
ManualAspectRatioWH	2
XAxisIndex	0
YAxisIndex	0
AutoYFit	
Enabled	False
MarginPercents	5
TargetAllYAxes	False
Thorough	True
UpdateInterval	100
AxisMouseWheelAction	Pan
CtrlEnabled	True
IgnoreZerosInLogFit	False
LeftMouseButtonAction	Zoom
MousePanThreshold	5
MouseWheelZooming	HorizontalAndVertical
MultiTouchPanEnabled	True
MultiTouchSensitivity	2
MultiTouchZoomDirection	Rails
MultiTouchZoomEnabled	True
PanDirection	Both
RectangleZoomAboutOrigin	False
RectangleZoomDirection	Both
RectangleZoomingThreshold	
X	4
Y	4
RectangleZoomLimitInsideGraph	False
RectangleZoomUnitsLinkYAxes	False
RightMouseButtonAction	Pan
RightToLeftZoomAction	FitView
ShiftEnabled	True
ViewFitYMarginPixels	0
ZoomFactor	2
ZoomOutRectFill	
ZoomOutRectLine	
ZoomRectFill	
ZoomRectLine	

Figure 5-91. ZoomPanOptions properties and sub-properties.

Zooming and panning are configurable and can be performed by left or right mouse button. Zooming can be also performed with mouse wheel.

5.22.1 Zooming with touch screen

Set two fingers on the chart and pinch the fingers closer to zoom out, or away to zoom in.

The chart tries to detect if trying to do a horizontal or vertical zooming, or both at same time. This feature is called 'zooming with rails', which can be controlled by **MultiTouchZoomDirection** (*Free/XAxis/YAxis/Rails*).

By pinching/spreading fingers above an X or Y axis or their labels, the zooming applies to that specific axis only.

Zooming with touch can be disabled by setting **MultiTouchZoomingEnabled = false**.

5.22.2 Panning with touch screen

Set two fingers on the screen and slide them at same pace to pan the view.

Some systems support panning with inertia, so it is possible to "throw" the fingers off the screen, and the view keeps panning and finally slows down until stopped.

By setting a finger above an X or Y axis or labels of it, and sliding the finger, the panning applies to that specific axis only.

5.22.3 Left mouse button action

Set **LeftMouseButtonAction** to **Zoom**, to enable zooming with left mouse button. Set it to Pan to enable panning. To disable zoom and pan from left mouse button, set it to **None**.

5.22.4 Right mouse button action

Set **RightMouseButtonAction** to **Zoom**, to enable zooming with right mouse button. Set it to Pan to enable panning. To disable zoom and pan from right mouse button, set it to **None**.

5.22.5 RightToLeftZoomAction

RightToLeftZoomAction applies when **LeftMouseButtonAction** or **RightMouseButtonAction** is set to **Zoom**. **RightToLeftZoomAction** specifies what happens when mouse zooming is made from right to left (mouse X button down-coordinate > button up-coordinate).

The following selections are available:

ZoomToFit: Fits all Y axes and X axis so that all series data belonging to them is shown. By using **ViewFitYMarginPixels** with greater value than 0, the axes are scaled so that given space in pixels is reserved empty of data, in both Y axis minimum and maximum end.

RectangleZoomIn: Zooms in with rectangle, as in zooming from left to right.

ZoomOut: Zooms out, by using **ZoomFactor**.

RevertAxisRanges: Sets axis values to specific values, which are restored after the view has been zoomed or axis ranges have been otherwise modified. In each axis, there's **RangeRevertEnabled** property, which controls if the axis range should be reverted. If it's enabled, **RangeRevertMinimum** and **RangeRevertMaximum** properties are applied to the axis when dragging mouse from right to left, and the mouse button is released.

PopFromZoomStack: Sets the same axis ranges that were used when zooming in last time, in other words, goes back to the previous zoom level.

5.22.6 Zooming with mouse button

5.22.6.1 Zoom in/out by clicking

Use **ZoomFactor** property to control the how much closer/farther the zoom is applied. To apply negative zoom effect, set value as inversed value (1/factor). The zoom is applied using mouse cursor position as a zoom center point.

X dimensional zoom:

With chart control focused, press Shift key down. Zoom X cursor appears. Click configured mouse button to zoom in, and the other button to zoom out.

Y dimensional zoom:

With chart control focused, press Ctrl key down. Zoom Y cursor appears. Click configured mouse button to zoom in, and the other button to zoom out. When using a stacked **YAxisLayout**, zooming applies to all graph segments (Y axes). By pressing Ctrl and Alt keys down, the Y dimensional zoom is applied only to the graph segment the mouse was clicked over.

Press both Shift and Ctrl(+Alt) keys down simultaneously, for applying zoom to both X and Y dimensions.

5.22.6.2 Zoom in with rectangle

With configured mouse button, drag a rectangle around the area to be zoomed, from upper left corner to bottom right corner. Both X and Y dimensions effect. The dimensions are selected by **RectangleZoomDirection** property. The zoom rectangle border and fill style can be modified by using **ZoomRectFill** and **ZoomRectLine** properties.

5.22.6.3 Configuring zoom out rectangle

When RightToLeftZoomAction is set to **ZoomToFit**, **ZoomOut**, **RevertAxisRanges** or **PopFromZoomStack**, the zoom out rectangle appears when zooming. Configure its fill by **ZoomOutRecFill** and line style by **ZoomOutRectLine**.

5.22.7 Zooming with mouse wheel

When **MouseWheelZooming** is enabled, zoom in by scrolling the mouse wheel upwards and zoom out by scrolling it downwards. The zoom center is the position of mouse cursor. Use **ZoomFactor** to adjust the mouse wheel zoom strength. By keeping Shift key pressed, the zoom is applied only to X dimension. By keeping Ctrl key pressed, the zoom applies only for Y dimension. Note that zooming is not available when **ScrollMode** is set to **Sweeping**.

5.22.8 Zooming and panning with mouse wheel, over axis

Use AxisMouseWheelAction to configure the outcome of mouse wheel actions applied over an axis.

None: Mouse wheel does nothing

Zoom: Zoom only the axis the mouse is over

Pan: Pan only the axis the mouse is over

ZoomAll: Zooms all X axes if mouse is over an X axis, or all Y axes is mouse over a Y axis. Applies to other axes only when **YAxisLayout = Layered**.

PanAll: Pans all X axes if mouse is over an X axis, or all Y axes is mouse over a Y axis. Applies to other axes only when **YAxisLayout = Layered**.

5.22.9 Panning with mouse button

Configure **LeftMouseButtonAction** or **RightMouseButtonAction** to **Pan** for panning to work. Drag the graph area with the configured mouse button pressed down. To stop panning, release the button. Panning scrolls both X and Y axes by dragged amount, if **PanDirection** is **Both**. By setting **PanDirection Vertical**, it only targets Y axes. Respectively, **PanDirection Horizontal** targets only X axes. Use **MousePanThreshold** to give some tolerance in pixels before the panning starts to affect. It's very handy when using ContextMenuStrip control assigned for the chart control, preventing it to open every time the panning stops.

5.22.10 Enabling/disabling Ctrl, Shift and Alt

Zoom operations support these modifier keys, and by default, they are enabled. To disable them, set **AltEnabled = False**, **CtrlEnabled = False** or **ShiftEnabled = False**.

5.22.11 Zoom in/out with code

Use **ZoomByFactor(...)** method to zoom with a center point and a zoom factor. Use **Zoom(...)** method to zoom with rectangle. **ZoomToFit()** method fits invokes "Zoom to fit" operation (fits all Y axes and X axis so that all series data is shown).

5.22.12 Zooming an axis by code

Set values to X or Y axis **Minimum** and **Maximum** properties. Use **SetRange(...)** to set them both at same time.

5.22.13 Rectangle zooming about a configurable origin

By enabling **RectangleZoomAboutOrigin**, the rectangle zooming in/out applies symmetrically using **ZoomOrigin** as a center point, set in X axis and Y axis values.



Figure 5-92. `ZoomPanOptions.RectangleZoomAboutOrigin` enabled. `ViewXY.XAxes[0].ZoomOrigin = 10` and `ViewXY.YAxes[0].ZoomOrigin = 50`.

5.22.14 Linking Y axes zoom with same units

By enabling `RectangleZoomLinkYAxes`, all the Y axes having the same `Units.Text` string get the same Y axis range as the axis that was rectangle zoomed.

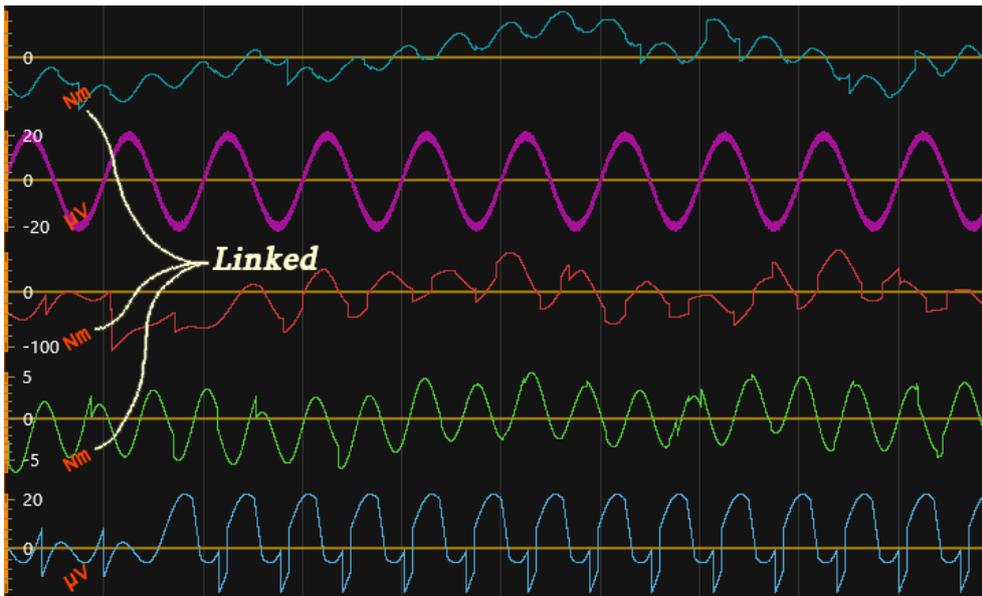


Figure 5-93. Stacked view with 5 Y axes. When rectangle zoom is applied over a graph segment, the Y axes of it get zoomed, and the new Y axis range is copied to all Y axes having the same `Units.Text`.

5.22.15 Automatic Y fit

Use **AutoYFit** property to control the automatic Y axis adjustment. Automatic Y fit can be used to adjust the Y axis ranges to show all the data in the chart in visible X axis range. Automatic Y fit is intended especially for real-time monitoring purposes. The fit is applied in time intervals, use **UpdateInterval** to set the interval in milliseconds. Use **MarginPercents** to get some extra range for axes. By enabling **Through**, the fitting analysis is made for all data, but may cause some overhead in performance critical systems. By disabling it, only a small piece of latest data is used for fitting routine and may cause improper behavior in certain applications.

Note! **AxisY** class also has **Fit()** methods for fitting in Y dimension.

5.22.16 Aspect ratio

AspectRatioOptions.AspectRatio controls the X/Y (or longitude / latitude in maps) ratio.

By default, it is set **Off** allowing X and Y axis ranges be set individually. By setting the aspect ratio to **Manual**, the **ManualAspectRatioWH** property can be used to set the preferred ratio. Changing **ManualAspectRatioWH** adjusts the x axis **Minimum** and **Maximum** properties to get the desired aspect ratio. Zooming operations will follow aspect ratio setting.

ManualAspectRatioWH is calculated as follows:

ManualAspectRatioWH = View width in pixels / View height in pixels * X axis range / Y axis range

For example:

ManualAspectRatioWH = 1530 / 902 * (20 - 0) / (100 - 0)

Width and height of the view depends on the window size. Axis ranges are simply maximum – minimum.

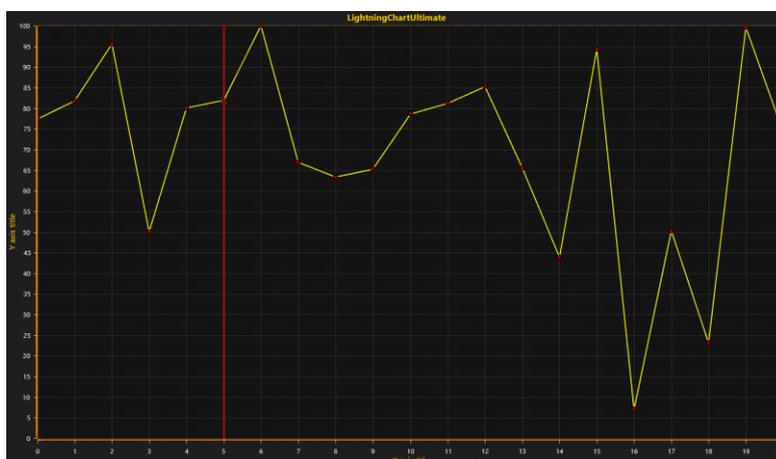


Figure 5-94. The view area of the chart. Its size in pixels is used to calculate the **ManualAspectRatioWH**.

When **AspectRatio** is other than **Off**, axis scaling nibs are not available.

For maps (see 5.25), **AspectRatio = AutoLatitude** is a very useful option. **AutoLatitude** changes the aspect ratio dynamically when viewing the map in different locations. The aspect ratio is determined by the center point of the view.

5.22.17 Excluding specific X or Y axes from zooming and panning operations

- To exclude specific X or Y axes from Zooming operations, set

axis.ZoomingEnabled = False

- To exclude specific X or Y axes from Panning operations, set

axis.PanningEnabled = False

5.23 DataBreaking by NaN or other value

DataBreaking	
Enabled	True
Value	NaN

Figure 5-95. DataBreaking options in series that support it.

These series types support data breaking:

- PointLineSeries
- FreeformPointLineSeries
- SampleDataSeries
- AreaSeries
- HighLowSeries
- PointLineSeries3D

LightningChart skips rendering of the data points that match with specified breaking Value. All other values it renders normally.

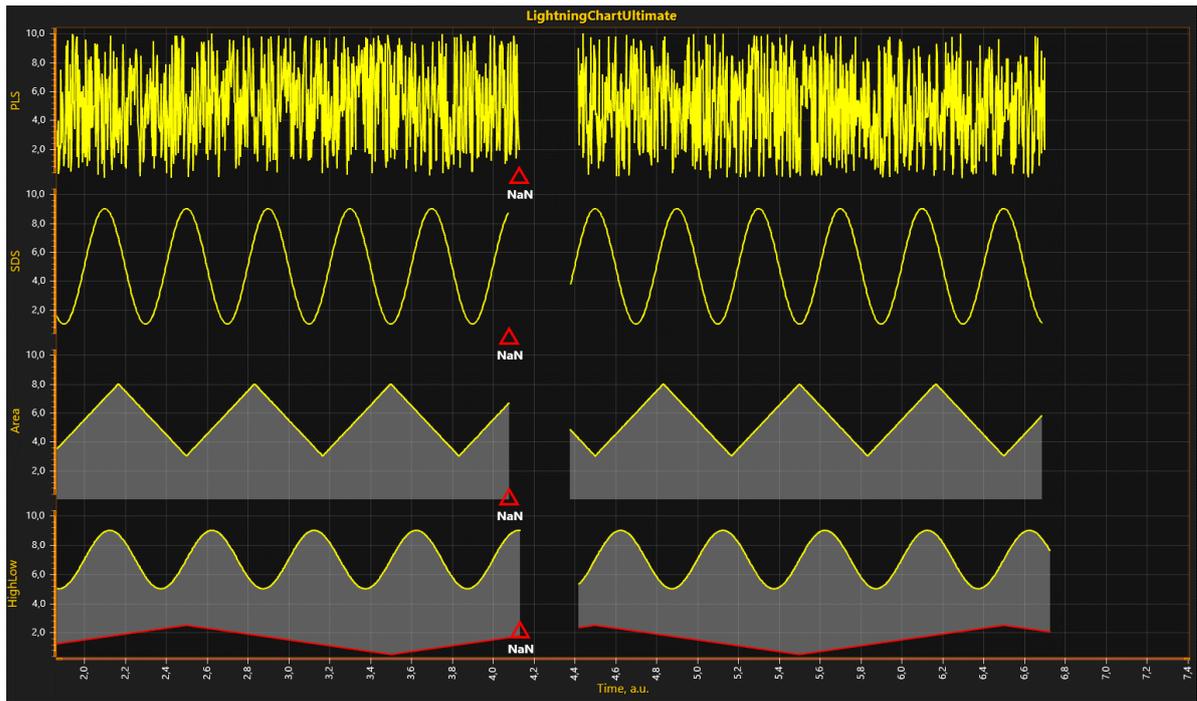


Figure 5-96. DataBreaking in use for PointLineSeries, SampleDataSeries, AreaSeries and HighLowSeries.

Note! When `DataBreaking.Enabled = True`, it will cause significant extra overhead, and is not recommended for solutions needing very high real-time data rates. Consider using `ClipAreas`, see chapter 5.24.

For example, using `NaN` to break `PointLineSeries` data:

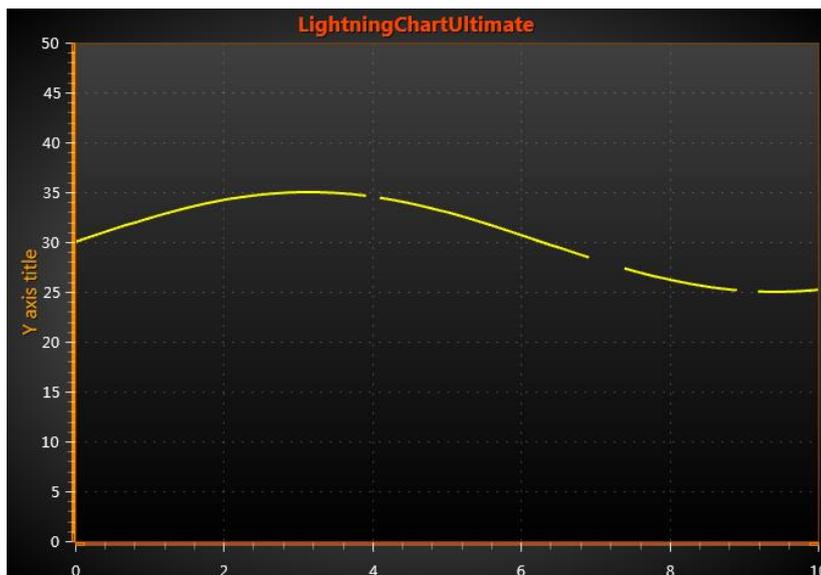


Figure 5-97. Using `NaN` to break `PointLineSeries`.

Code:

```
int pointCount = 101;
double[] xValues = new double[pointCount];
double[] yValues = new double[pointCount];

for (int point = 0; point < pointCount; point++)
{
    xValues[point] = (double)point * interval;
    yValues[point] = 30.0 + 5.0 * Math.Sin((double)point / 20.0);
}

//Add some NaN values in Y array to mark break points
yValues[40] = double.NaN;
yValues[70] = double.NaN;
yValues[71] = double.NaN;
yValues[72] = double.NaN;
yValues[73] = double.NaN;
yValues[90] = double.NaN;
yValues[91] = double.NaN;

//Add new series with DataBreaking Enabled
PointLineSeries pls = new PointLineSeries(_chart.ViewXY,
_chart.ViewXY.XAxes[0], _chart.ViewXY.YAxes[0]);
pls.DataBreaking.Enabled = true;

// set data gap defining value (default = NaN)
pls.DataBreaking.Value = double.NaN;

SeriesPoint[] points = new SeriesPoint[pointCount];
for (int point = 0; point < pointCount; point++)
{
    points[point].X = xValues[point];
    points[point].Y = yValues[point];
}

//Assign the data for the point line series
pls.Points = points;

//Add the created point line series into PointLineSeries list
_chart.ViewXY.PointLineSeries.Add(pls);
```

5.24 ClipAreas

Like **DataBreaking** (see 5.23), **ClipAreas** can be used to prevent part of the series data from rendering. They can be used to filter out bad data ranges, out-of-range data by Y value, etc.

ViewXY's series have **SetClipAreas** method for setting or updating the clipping areas. It accepts an array of **ClipArea** structures. The ClipAreas array can be changed frequently, and performance stays good up to thousands of ClipAreas.

The **ClipArea** applies for the series that it has been assigned to. Note that this is a rendering-stage clipping and mouse operations will respond to series when placed over the ClipArea if there's actual data under it.

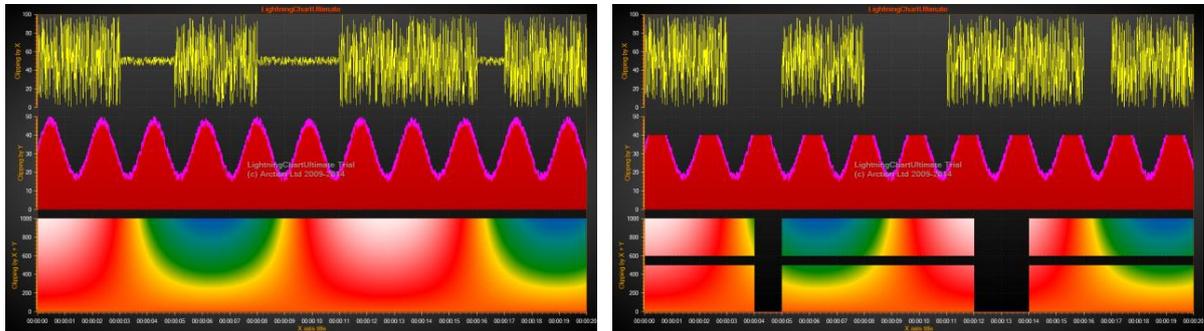


Figure 5-98. ClipAreas defined for 3 series. For PointLineSeries, AreaSeries, and IntensityGridSeries. On the left, the ClipAreas are not used. On the right, ClipAreas are enabled. For yellow PointLineSeries, X dimensional clipping areas have been defined to mask off low-amplitude data. For red AreaSeries, Y-dimensional ClipArea cuts too high-amplitude data from the top. For IntensityGridSeries, X- and Y-dimensional ClipAreas are used to prevent the series from rendering in specific areas.

Using ClipAreas is the performance-wise preferred way to break a line to several data segments instead of using DataBreaking feature, or spawning hundreds of separate series during real time monitoring.

5.25 Maps

Use **Maps** property and its sub-properties to show geographic maps. LightningChart maps come in two different categories: **vector maps** and **tile maps**. The maps are shown in so called *equiarectangular* projection.

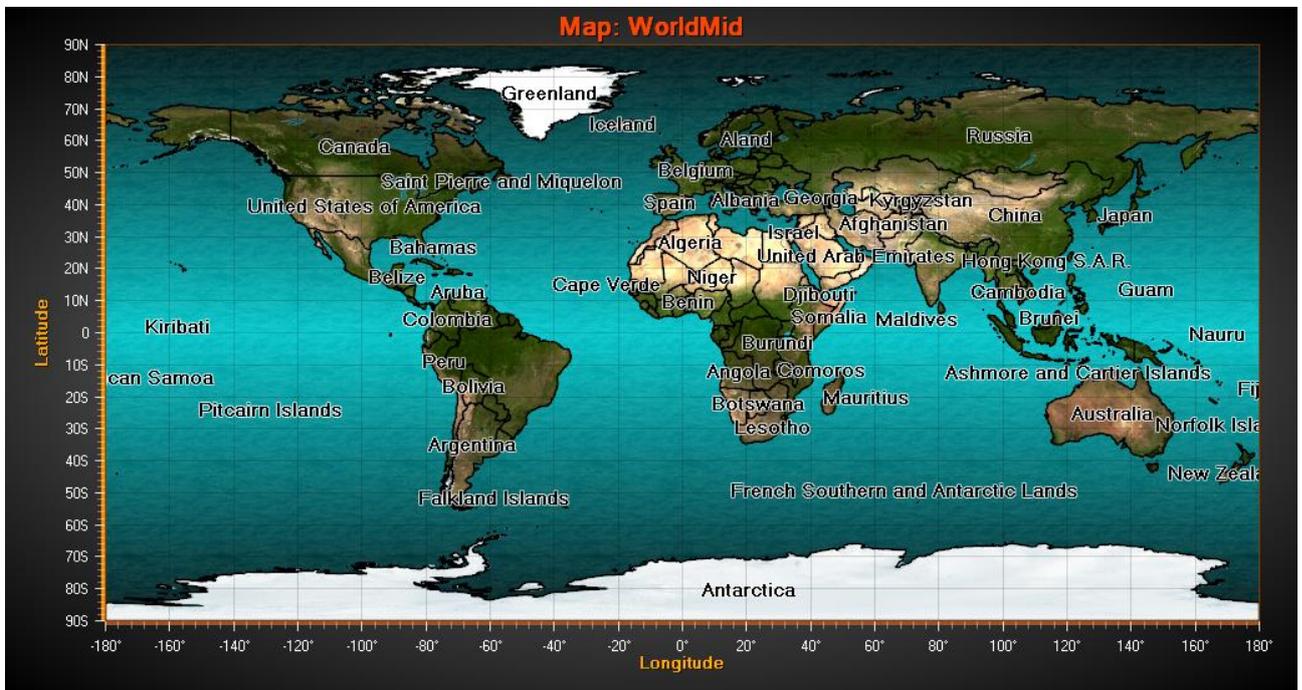


Figure 5-99. Equiarectangular projection of the world. X range is from -180 to 180 degrees (180W to 180E), Y range from -90 to 90 degrees (90S to 90N). Polar areas get greatly stretched in this projection.

This projection allows using LightningChart's series types and other objects that are practically all bound to X and Y axes, same time with the maps.

5.26 Vector maps

The geographic vector data is stored in LightningChart map files, with **.md** extension. LightningChart is delivered with set of map files.

The X Axis is used for Longitude, and the Y axis for latitude. See chapter 5.2.3 for showing map coordinate axes. The map coordinates are decimal degrees, with latitude origin at equator and longitude origin at Greenwich, U.K.

[-] Maps	
Backgrounds	(Collection)
[-] CityOptions	
Description	Map of world in mid resolution.
FileName	WorldMid
[-] LakeOptions	
[-] LandOptions	
[-] Layers	MapLayer[] Array
MouseHighlight	Simple
MouseInteraction	True
MouseOverMapItemLayer	1
Names	(Collection)
Optimization	None
[-] OtherOptions	
OverlapLabels	False
Path	..\..\Maps
RenderIntensitySeriesBeforeLayerIndex	-1
[-] RiverOptions	
[-] RoadOptions	
SimpleHighlightColor	 100, 0, 255, 0
TileCacheFolder	c:\temp\map_cache
TileLayers	(Collection)
Type	WorldMid
XAxisIndex	0
YAxisIndex	0

Figure 5-100. Maps' properties and sub-properties. The whole tree is for vector maps, except TileLayers collection and TileCacheFolder, which is for tile maps.

5.26.1 Selecting active map

Set the directory name to the **Path** property, where the map files exist. The active map can be selected with **Type** property, for maps delivered with LightningChart. To use an own map file, set the **FileName** property.

If no maps are wanted, set **Type** to **Off**.

Off
AustraliaMid
CanadaUSASatesMid
EuropeLow
EuropeMid
EuropeHigh
Other
USALakesRiversMid
USALakesRiversHigh
USASatesLakesRiversMid
USASatesLakesRiversHigh
USASatesLakesRiversRoadsMid
WorldLow
WorldMid
WorldHigh
WorldLakesRiversLow
WorldLakesRiversMid
NorthAmericaLow
NorthAmericaMid
NorthAmericaHigh

Figure 5-101. Map Type options. The maps delivered with LightningChart are shown. The type name postfix tells a rough detail level of the map.

In general, the maps of LightningChart are made in very high detail level. For real-time monitoring solutions it is important to select a map giving proper detail and performance level.

5.26.2 Aspect ratio

ViewXY.ZoomPanOptions.AspectRatioOptions.AspectRatio controls the X/Y (or longitude / latitude) ratio.

Set it to ***Off*** to enable X and Y axis range setting individually allowing stretching the map. ***AutoLatitude*** changes the aspect ratio dynamically when viewing the map in different locations. The aspect ratio is determined by the center point of the view. By setting the aspect ratio to ***Manual***, use the ***ManualAspectRatioWH*** property to set the preferred ratio. See chapter **Error! Reference source not found.** for detailed explanation of how aspect ratio is calculated.

5.26.3 Layers and their appearance settings

Each map file can contain several layers. For example, layers for land regions, lakes, rivers, roads and cities. The layers and their data are accessible from **Layers** array property.

Layers	MapLayer[] Array
[0]	Arction.LightningChartUltimate.Maps.Poir
Color	
Items	City[] Array
Name	Cities
Priority	0
Type	City
Visible	True
[1]	Arction.LightningChartUltimate.Maps.Reg
[2]	Arction.LightningChartUltimate.Maps.Reg
BorderDrawStyle	Options
Items	Region[] Array
Name	Lakes
Priority	2
RegionDrawStyle	Options
Type	Lake
Visible	True
[3]	Arction.LightningChartUltimate.Maps.Line
AutoAdjustLineWidth	True
Items	Line[] Array
LineDrawStyle	Options
LineWidthCoeff	1
Name	Rivers2
Priority	3
Type	River
Visible	True

Figure 5-102. Map layer details opened in Properties editor.

Each layer has a specific type. The layer appearance options can be changed with corresponding options property. Use **LandOptions** for modifying the appearance of land regions, **LakeOptions** for lakes, **RiverOptions** for rivers, **RoadOptions** for roads, **CityOptions** for cities, and **OtherOptions** for unspecified layer types.

LandOptions	Arction.LightningChartUltimate.h
AntialiasFill	False
Fill	Arction.LightningChartUltimate.F
Bitmap	Arction.LightningChartUltimate.B
Color	OldLace
GradientColor	Moccasin
GradientDirection	270
GradientFill	Linear
Style	ColorOnly
FillVisible	True
LabelStyle	Arction.LightningChartUltimate.h
Angle	0
Color	Black
Font	Microsoft Sans Serif; 12pt; style
Shadow	Arction.LightningChartUltimate.T
Visible	True
LineStyle	Arction.LightningChartUltimate.L
AntiAliasing	Normal
Color	DimGray
Pattern	Solid
PatternScale	1
Width	1
LineVisible	True

Figure 5-103. Default LandOptions, and corresponding view from Europe.

LandOptions	Arction.LightningChartUltimate.M
AntialiasFill	False
Fill	Arction.LightningChartUltimate.Fi
Bitmap	Arction.LightningChartUltimate.B
Color	SandyBrown
GradientColor	Black
GradientDirection	270
GradientFill	Radial
Style	ColorOnly
FillVisible	True
LabelStyle	Arction.LightningChartUltimate.M
Angle	45
Color	Black
Font	Arial Black; 12pt; style=Bold, Ital
Shadow	Arction.LightningChartUltimate.T
Visible	True
LineStyle	Arction.LightningChartUltimate.Li
AntiAliasing	Normal
Color	Black
Pattern	Solid
PatternScale	1
Width	3
LineVisible	True



Figure 5-104. Modified LandOptions.

5.25.3.1 Setting individual fill and border style for each layer item

Each map element fill or border appearance can be set individually. Change **BorderDrawStyle** and **RegionDrawStyle** properties to **Individual**. Then, access the Items collection, and navigate to preferred item and edit the **BorderLineStyle** and **Fill** properties. The **Items** collection can be navigated programmatically by **Name** property, here “Germany”.

Layers	MapLayer[] Array
[0]	Arction.LightningChartUltimate.Maps.Pc
[1]	Arction.LightningChartUltimate.Maps.Re
BorderDrawStyle	Individual
Items	Region[] Array
Name	Countries
Priority	1
RegionDrawStyle	Individual
Type	Land
Visible	True

[54]	Arction.LightningChartUltimate.Maps.R
[55]	Arction.LightningChartUltimate.Maps.R
[56]	Arction.LightningChartUltimate.Maps.R
[57]	Arction.LightningChartUltimate.Maps.R
BorderLineStyle	Arction.LightningChartUltimate.L
AntiAliasing	None
Color	Red
Pattern	Solid
PatternScale	1
Width	3
Center	Arction.LightningChartUltimate.PointFc
Fill	Arction.LightningChartUltimate.F
Bitmap	Arction.LightningChartUltimate.B
Color	White
GradientColor	DarkViolet
GradientDirec	270
GradientFill	Linear
Style	ColorOnly
Name	Germany

Figure 5-105. Setting layer border line and region fill styles to Individual and editing region in Items collection.

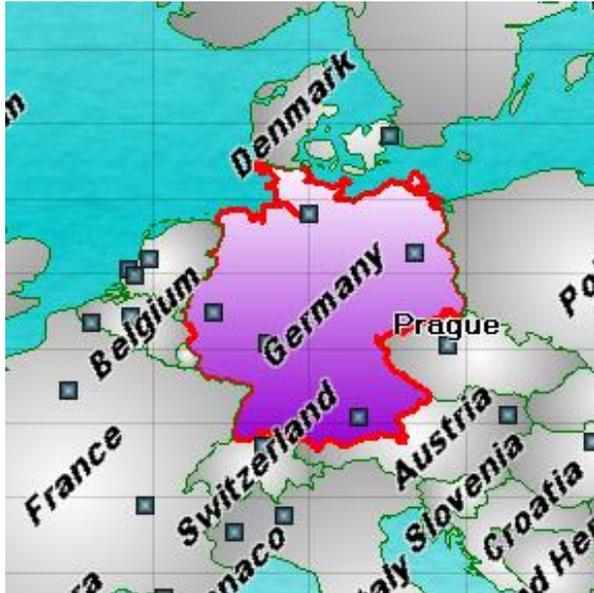


Figure 5-106. Germany region drawn with individual fill and border.

5.26.4 Mouse interactivity

Enable ***MouseInteraction*** for all kind of interoperation with map regions and objects. Regions (land, lakes) and vector layers (rivers, roads) can be pointed with mouse. Once the mouse is over an object, it gets highlighted with ***SimpleHighlightColor***, if ***MouseHighlight*** is set to ***Simple***. When ***MouseHighlight*** is set to ***Blink***, the object will blink in light and dark colors. By setting ***MouseHighlight*** to ***None***, the object is not highlighted, but it still can be clicked and for example used to invoke ***Maps.MouseDownOnMapItem*** event.

Map objects may have associated data included, like population or other statistical data. Use ***MouseOverOnMapItem/MouseOverOffMapItem/MouseDownOnMapItem*** event handler to access the data. The data for a map item can be retrieved with ***GetInfo*** method, giving a dictionary of keys and values.

Here's an example of how to show all data in a list box. The item name is displayed in a different text box.

```

private void MouseDownOnMap(MouseDownOnMapItemEventArgs args)
{
    MapItem mapItem = args.MapItem;

    textBoxCountryName.Text =
        m_chart.ViewXY.Maps.Layers[args.Layer].Name
        + ": " + mapItem.Name;
    listBoxItemValues.Items.Clear();
    if (mapItem.GetInfo() != null)
    {
        Dictionary<string, string> dict = mapItem.GetInfo();
        Dictionary<string, string>.KeyCollection keys = dict.Keys;
        foreach (String key in keys)
        {
            String strValue;
            if (dict.TryGetValue(key, out strValue))
            {
                listBoxItemValues.Items.Add(key + ": " + strValue);
            }
        }
    }
}

```

5.26.5 Background photos

Adding a **MapBackground** object in **Maps.Backgrounds** property allows displaying bitmap images as the backgrounds of the maps. Satellite images or other raster images are available from several GIS data providers. The image can be set to **Image** property, and its latitude and longitude range can be set with **LatitudeMin**, **LatitudeMax**, **LongitudeMin** and **LongitudeMax** properties. The image is not displayed outside the set ranges.

To show the background through the map layers, it may be necessary to adjust the fill settings for each layer. Use transparent colors or colors with low alpha level.

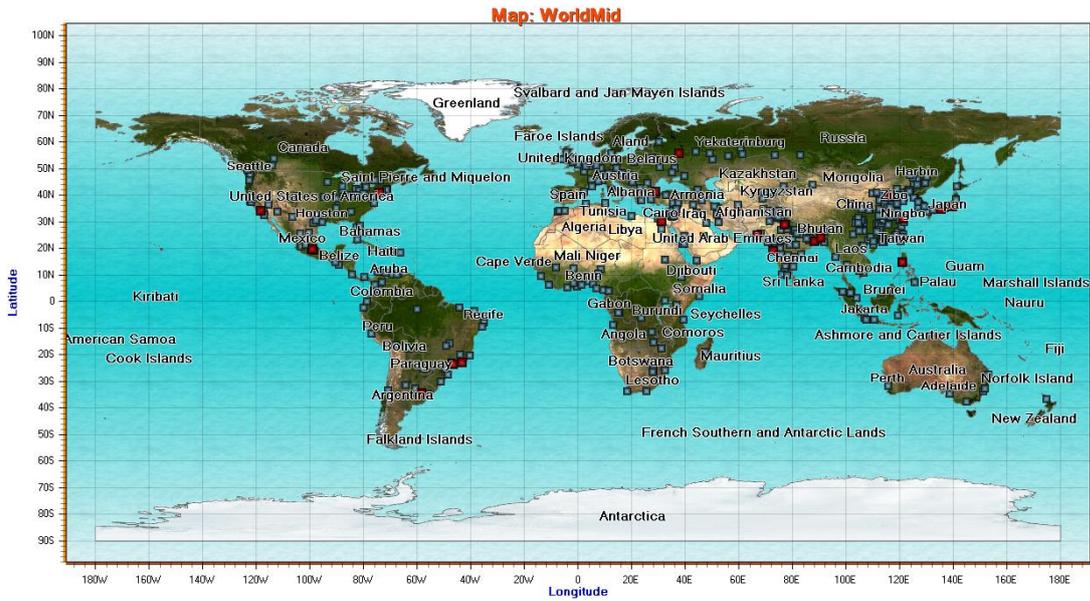


Figure 5-107. Map of the world. LandOptions.FillVisible is set to false, and one background image is set to latitude range of -90...90 and longitude range of -180...180. The map region borders and cities are shown.

5.26.6 Combining other series with maps

Geographical maps can be combined with any ViewXY series type. The maps are drawn in the background and the series over them.



Figure 5-108. Map of Europe, with a couple of FreeformPointLine series as routes. Flag markers are added to them as mouse-interactive waypoints.



Figure 5-109. Map of the world, with IntensityGrid series presenting the elevation.

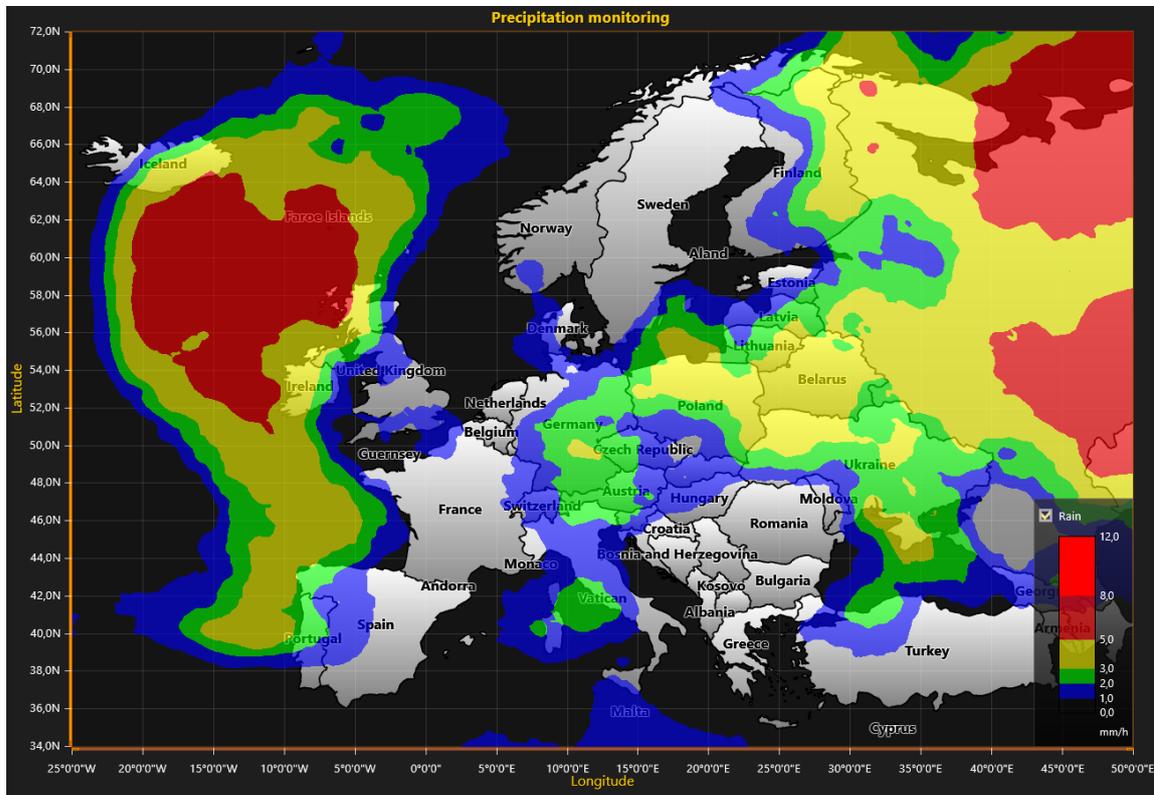


Figure 5-110. Weather radar data visualization with IntensityGrid series over the map of Europe.

5.26.7 Importing maps from ESRI shape file data

Import feature makes a LightningChart map file (.md) from .shp files. ESRI shapefile (*.shp) is a widely used map file format supporting vector and polygon data.

Map wizard can be used to convert shapefile data to LightningChart (LC) map data format. LC format supports layering, so multiple shapefiles can be merged into a single file. Map file structures and objects are pre-processed for maximum run-time performance.

Tip: *LightningChart Ultimate demo application has an example of map importing. Run import wizard from there to make custom LC map files through import.*

Conversion is done in minimum of three steps:

1. Selecting files and setting up layers based on the files in the Shapefile Selection Dialog.
2. Determining file text encoding.
3. Selecting items included in the resulting map file.

Note that steps 2 and 3 are repeated for each source shp file. Shapefile does not tell which encoding it uses, so it must be selected by the user.

After the steps, the conversion begins. If the maps are imported from a custom application, the developer is encouraged to setup an event handler, because conversion might take a very long time, so that user can be informed about the conversion progress.

Also, if user selects base layer, there might be a considerable delay between the steps, which is due to prefiltering data based on the layer.

5.26.7.1 Programming interface for importing shp data

Conversion is run on a thread that is initialized from Maps.MapConverter class using following method:

```
public bool SelectFilesAndConvert()
```

For monitoring conversion progress there is an event handler delegate:

```
public delegate void ConversionStateChangedHandler(ConversionProgress progress, int i);
```

Initializing it:

```
MapConverter mapConverter = new MapConverter();  
mapConverter.ConversionStateChanged += new  
MapConverter.ConversionStateChangedHandler(mapConverter_ConversionStateCh  
anged);
```

5.26.7.2 Dialogs

There are usually three dialogs involved in the conversion process. For selecting a filter, there is a distinct dialog.

5.26.7.2.1 Shapefile Selection Dialog

After `SelectFilesAndConvert()` function is called, file selection dialog opens. In this dialog user selects the source files and sets up the layering. User can also save the map configuration by selecting proper file at the dialog.

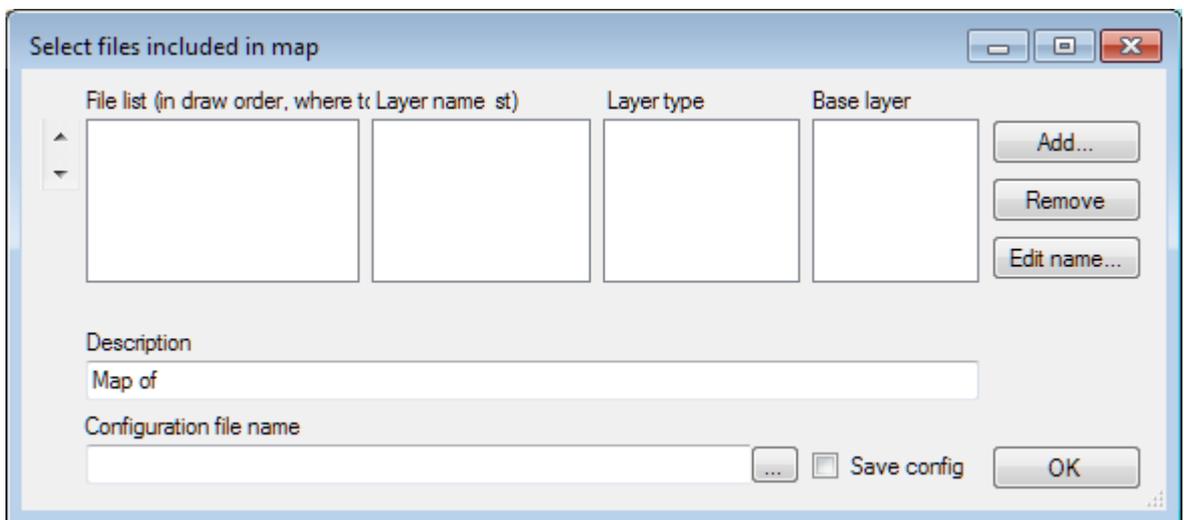


Figure 5-111. Source shape file selection dialog.

File list

Contains list of files in the order in which they are drawn. File data at bottom is drawn last. Order of files can be changed from the up/down buttons on the left of list. Select file and click up/down to move file.

Layer name

Name of the layer. E.g. "Countries".

Layer type

Type of layer (specifies which options are used to render layer)

- City: layer items are of shapefile type POINT
- Lake: layer items are of shapefile type POLYGON

- Land: layer items are of shapefile type POLYGON
- River: layer items are of shapefile type POLYLINE
- Road: layer items are of shapefile type POLYLINE
- Other: layer items are of shapefile type POLYGON or POLYLINE

Base layer

Used to filter upper layer item selection, when user wants a map which contains only single/some countries and there is only global map available. E.g. if layer contains countries, only items over the selected countries/country will be included in the resulting map. There is a small offset applied to POINT type, so that if point is near enough of border it's included even if it doesn't overlap with base layer. *If all data from the selected shapefiles are included in the resulting map, don't select base layer as it slows down item selection considerably, because all items are checked if they overlap base layer, which is a very time consuming process.*

Description

Free text which is shown in the map properties.

Configuration file name

XML configuration file name. Used for importing/replacing a layer. **Note!** Use single file when creating map configuration as import. Replace methods can take only one shp input file.

Save config

Check this if wanting to save map configuration as xml file for later use. Selecting configuration file automatically sets this checked.

Add button

Click to select shapefile to be added to list.

Remove button

Removes selected file from list.

Edit name button

Click to open "Layer name editor". Set layer name.

OK button

Click to advance to next stage (item selection).

5.26.7.2.2 Select Record Encoding and Invalid Name Fields

This dialog is used to select file text encoding and fields which have invalid or general name. Shape file encoding may vary and there is no information about the encoding in the file, so user must select valid encoding. The item name may be like "UNK" for multiple items. In this dialog

the user can select which item name is emptied. Note that the items are still included in the resulting file, if they are selected in the next phase.

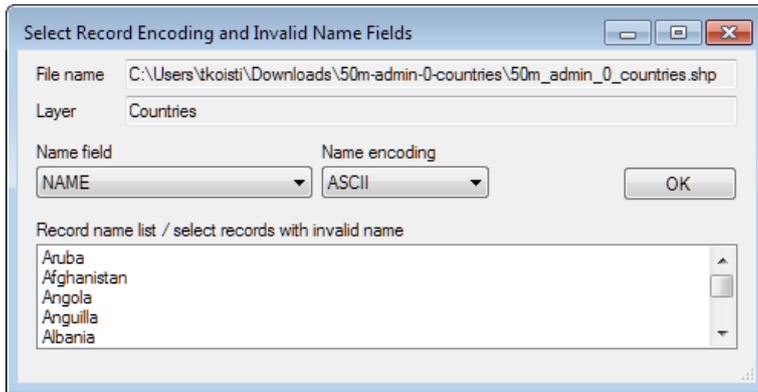


Figure 5-112. 'Record encoding' and 'Invalid name' fields selection dialog.

File name

Shape file name for which the encoding applies.

Layer

Layer name.

Name field

Item name field in the shape file. After selecting a different field, the list is updated accordingly.

Name encoding

Item name encoding (try different values if the name does not seem to be right). After selecting different encoding, the list is updated accordingly.

Record name list / select records with invalid name

List of items for the field selected in the "Name" field.

OK

Confirm encoding selection (and possible invalid name).

5.26.7.2.3 Layer data selection dialog

This dialog is used to select items included in the resulting map file from the shape file. The layer name is concatenated in the title. The dialog is adaptive, so that for certain layers there are some fields which could be selected. E.g. for **River/Road** type layer there will be a Line width selection, which could be set to line width field (if applicable). Note that the data may not contain all the fields asked in the dialog. The **Name** field is mandatory for all items.

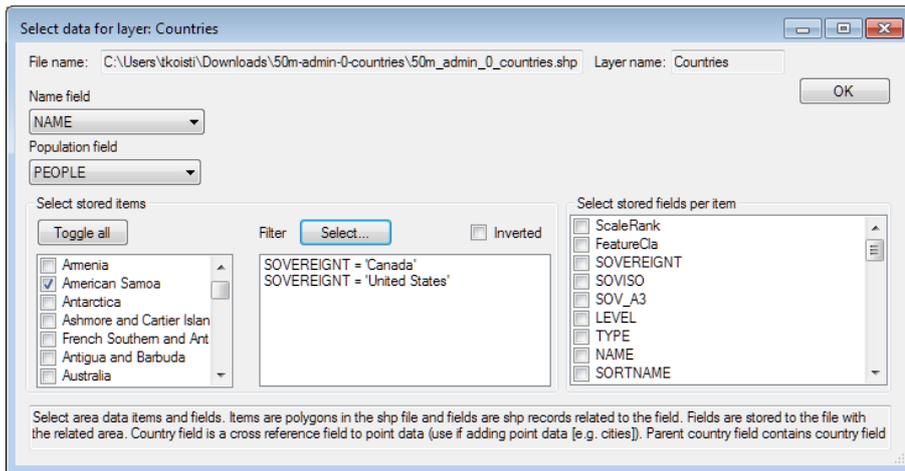


Figure 5-113. Layer data selection dialog.

The user interface items available in the dialog:

File name

Name of the file.

Layer name

Name of the layer.

Name field

Field used for item name. Set automatically from encoding selection dialog but can be adjusted here also.

Population field

Field used for population data.

Country field

Country name field.

Line stroke width field

Line width. Guides rendering of lines.

Select stored items

Select items individually, select them all, or use **Filter** dialog to select subset of items

Toggle all

Select all fields from file.

Filter/Select...

Select fields which has a field with selected values. In the image above, only items with SOVEREIGNT field set to "Canada" or "United States" are selected in the map.

Inverted

Invert filter selection (fields selected with filter are not included in resulting map file).

Select stored fields per item

Click on fields which should be included for each item. The fields are key values for Dictionary class, which contains the fields per item.

5.26.7.2.4 Item filter

This dialog is opened from Layer data selection dialog and is used to filter items for resulting map.

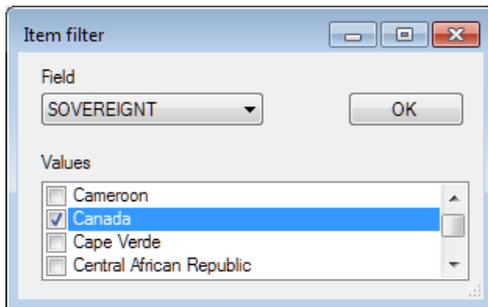


Figure 5-114. Item filter dialog.

Field

Select the field on which the filtering is based.

Values

Select values which are included in the resulting items.

The above selection means that items, whose field name SOVEREIGNT contains value "Canada", are included in the resulting map.

5.26.8 Importing and replacing map layers

User can import new layers to the map and replace existing layers. There are four methods for importing and replacing a layer in the map from Maps interface. This is very useful when retrieving frequently updated shp data while the software application is running.

ImportNewLayer methods inserts a new map layer to given layer index and **ImportReplaceLayer** methods replaces the map layer at the given layer index.

```
public MapConverter.ConversionResult ImportNewLayer(String shpFilename,  
int targetLayerIndex),
```

where **shpFilename** is the name of the source shp file name and **targetLayerIndex** is the index of the new layer. This method uses dialogs presented above for setting up map configuration.

```
public MapConverter.ConversionResult ImportNewLayer(String shpFilename,  
int targetLayerIndex, String configFile),
```

where **shpFilename** is the name of the source shp file name, **targetLayerIndex** is the index of the new layer and **configFile** is map configuration file name. This method uses configuration file created with dialogs presented above.

```
public MapConverter.ConversionResult ImportReplaceLayer(String  
shpFilename, int targetLayerIndex),
```

where **shpFilename** is the name of the source shp file name and **targetLayerIndex** is the index of the new layer. This method uses dialogs presented above for setting up map configuration.

```
public MapConverter.ConversionResult ImportReplaceLayer(String  
shpFilename, int targetLayerIndex, String configFile),
```

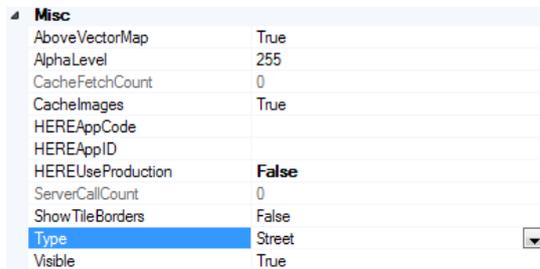
where **shpFilename** is the name of the source shp file name, **targetLayerIndex** is the index of the new layer and **configFile** is map configuration file name. This method uses configuration file created with dialogs presented above.

Configuration file is a plain xml file, which can be edited with a text editor, though editing is not recommended.

5.27 Tile maps

LightningChart has support for the following on-line tile data services:

- [Here](#): Street maps, Satellite imagery



Misc	
AboveVectorMap	True
AlphaLevel	255
CacheFetchCount	0
CacheImages	True
HEREAppCode	
HEREAppID	
HEREUseProduction	False
ServerCallCount	0
ShowTileBorders	False
Type	Street
Visible	True

Figure 5-115. Properties of a `TileLayer`.

Add ***TileLayer*** object(s) in **`ViewXY.Maps.TileLayers`** collection. Several layers can be inserted and made semi-transparent with ***AlphaLevel*** property. The ***TileLayer*** objects are rendered in the order of appearance in the ***TileLayers*** collection, the first layer being in the background. By setting ***AboveVectorMap*** = ***False***, the layer renders before the vector map, if such are defined (see 5.26). By default, the ***TileLayer*** renders after the vector maps.

TileLayer gets information as small images from on-line service provider through http protocol and shows them in the chart area. The images are refreshed when zooming or panning the map view.

Loading a new set of tiles will take some time, up to several seconds.

Tile cache

The chart stores the tiles into a cache folder, which greatly reduces the loading time when panning or zooming frequently in the same region. When the chart needs to show a tile, it first checks if it can be found in the cache folder, and if not, retrieves it from the web service. In a team use, where many workstations need to access the tile maps, it is wise to select a shared local network server folder. By default, the cache folder is **`c:\Users\[Current user]\AppData\Local\Temp`**.

Set the cache folder in **`ViewXY.Maps.TileCacheFolder`**.

Clear the cache folder by calling **`ViewXY.Maps.ClearTileCacheFolder()`** method.

5.27.1 HERE

LightningChart supports tile data service by Here. Developer or end user must make an own contract with Here to be able to use the Here servers. Free trial keys can be acquired from

<https://developer.here.com/plans/api/consumer-mapping>

Selecting type

Set **TileLayer.Type = Street** to use street maps. The street maps can be zoomed in very near.

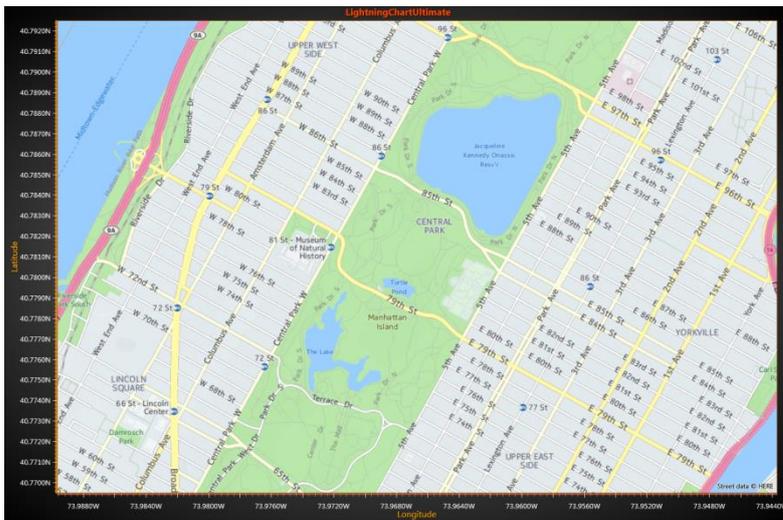


Figure 5-116. TileLayer.Type = Street.

Set **TileLayer.Type = Satellite** to use satellite imagery.

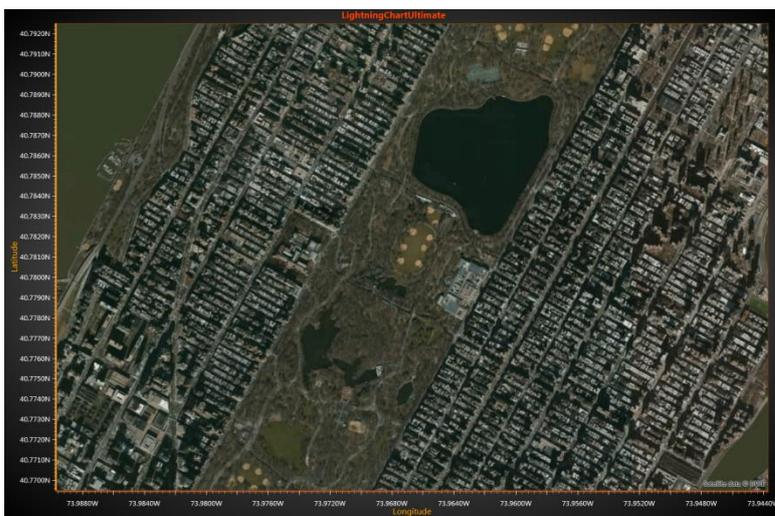


Figure 5-117. TileLayer.Type = Satellite.

Presenting series and other chart elements, like annotations, is possible.

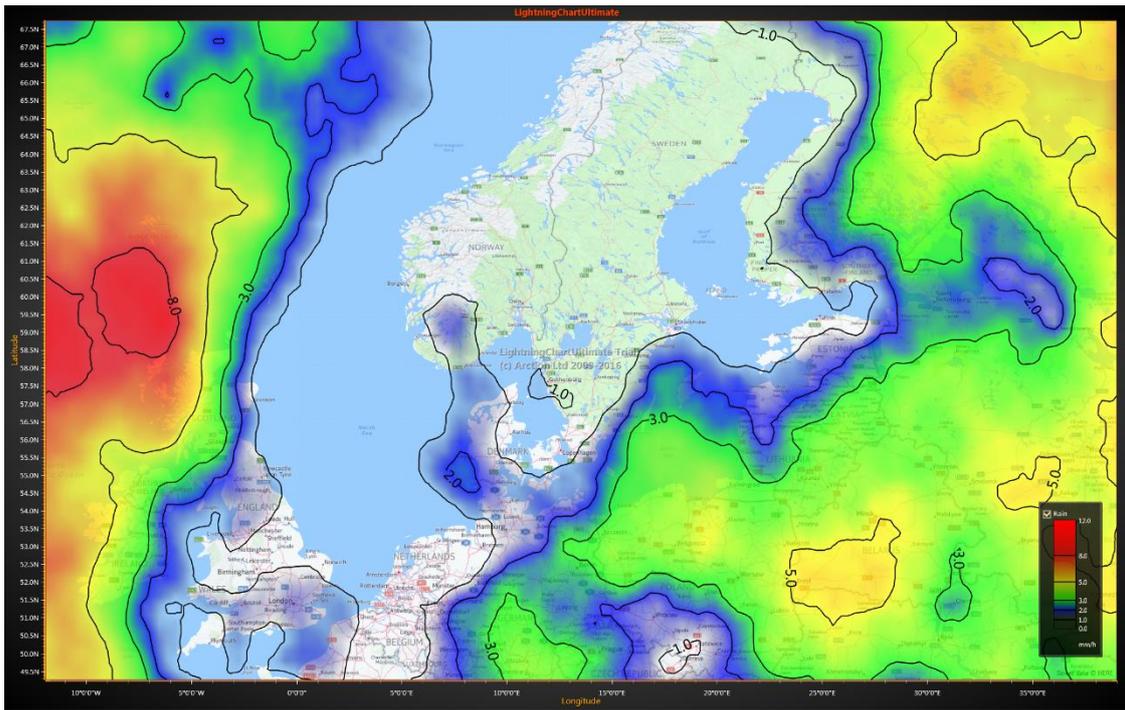


Figure 5-118. Street maps with IntensityGridSeries presenting weather data.

5.28 Stencil areas

IntensityGridSeries, *IntensityMeshSeries* and *Maps* have *StencilArea* feature. It allows masking in or out areas of drawn data. Create *StencilArea* objects and add them to the series that should be masked.

To create a positive stencil mask: `seriesOrMap.Stencil.AdditiveAreas.Add(createdStencilArea);`

To create a negative stencil mask: `seriesOrMap.Stencil.SubtractiveAreas.Add(createdStencilArea);`

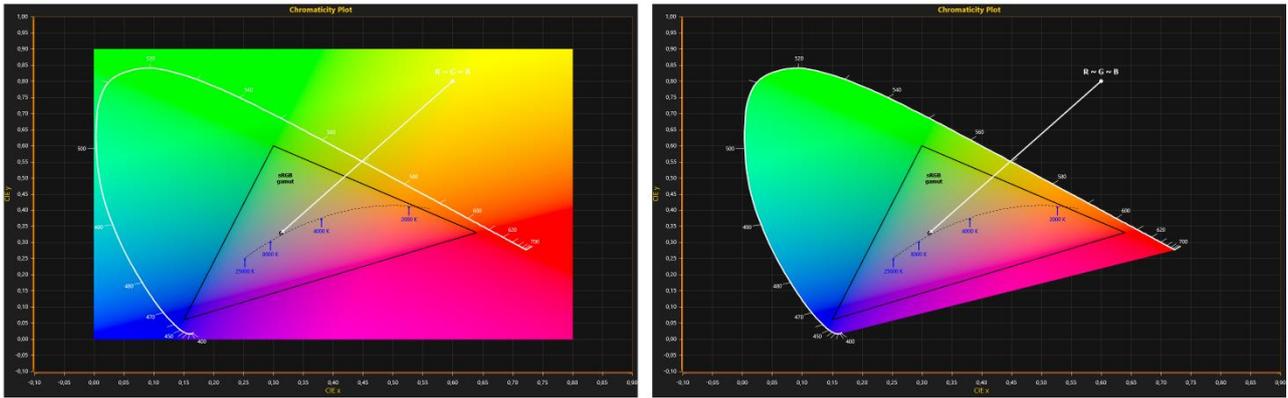


Figure 5-119. On the left, no StencilAreas are not used. On the right, a positive stencil mask as a polygon is added using `intensityGridSeries.Stencil.AdditiveAreas.Add()` -method.

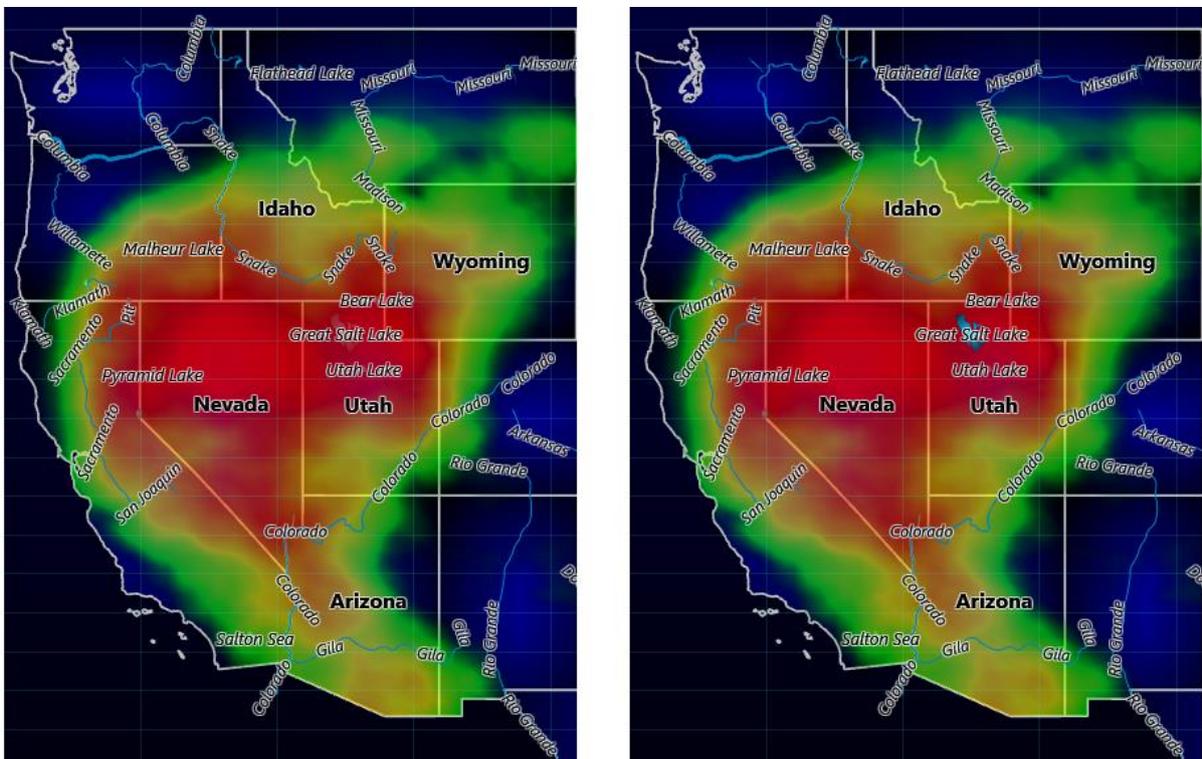


Figure 5-120. Environmental data is shown over a map. Each state is a positive stencil area. On the right, lakes are added as negative stencil masks using `intensityGridSeries.Stencil.SubtractiveAreas.Add()` -method. In this case data is not shown over the Great Salt Lake.

5.29 Line series cursors

Line series cursors allow visual analysis of line series data by tracking the values by X coordinate. Series values can only be resolved with series implementing *ITrackable* interface (*SampleDataSeries*, *PointLineSeries*, *AreaSeries*, *HighLowSeries*). For other series types, Y coordinate is not automatically tracked by cursors.

Add *LineSeriesCursor* object into *LineSeriesCursors* collection. Enable *SnapToPoints* to jump the cursor from point to point. Set the cursor tracking style with *Style* property. When *Style* is set to *PointTracking*, any tracking point style can be used, even a bitmap image. When using *HairCrossTracking* style, a horizontal line is drawn at line series point Y value. If multiple points of same series hit in cursor location, line is drawn in the middle of minimum and maximum points.

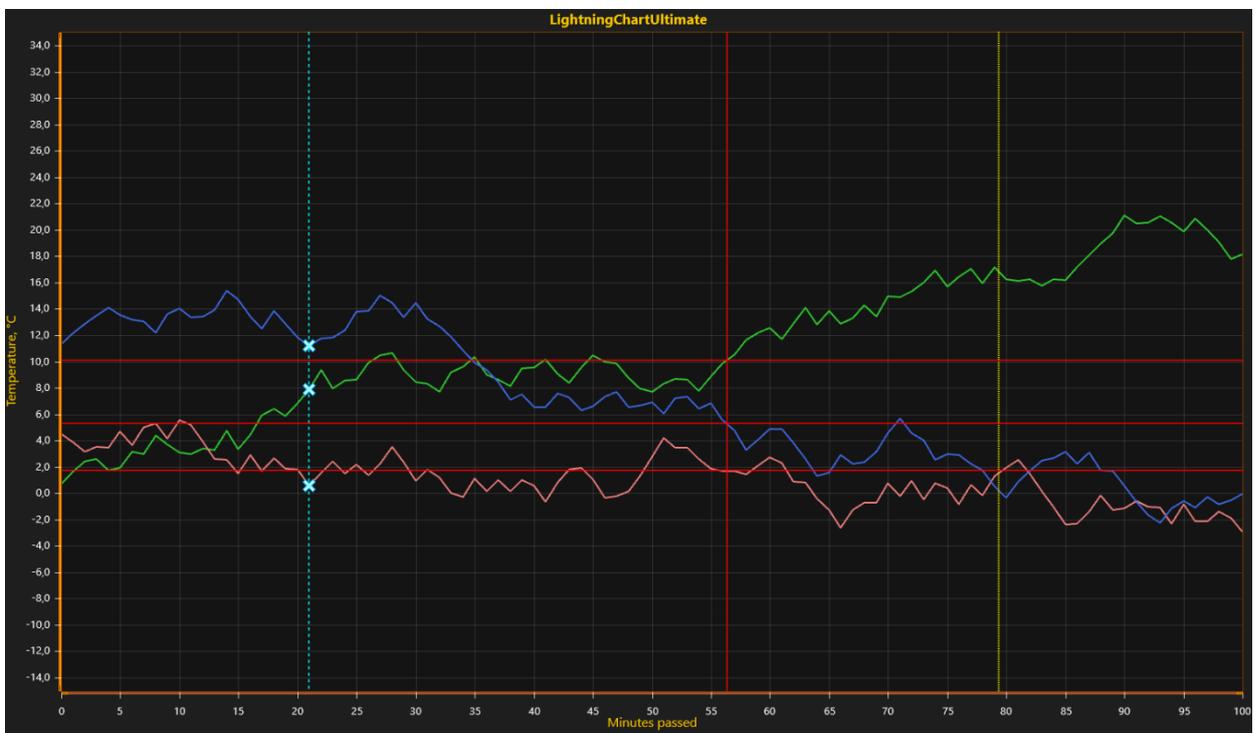


Figure 5-121. Line series cursors: Vertical point tracking cursor (cyan line and crosses), hair-cross tracking cursor (red lines) and vertical full-length cursor without tracking (yellow line)

By enabling *IndicateTrackingYRange* a horizontal bar is drawn ranging from minimum to maximum of the points hitting in the middle of the cursor.

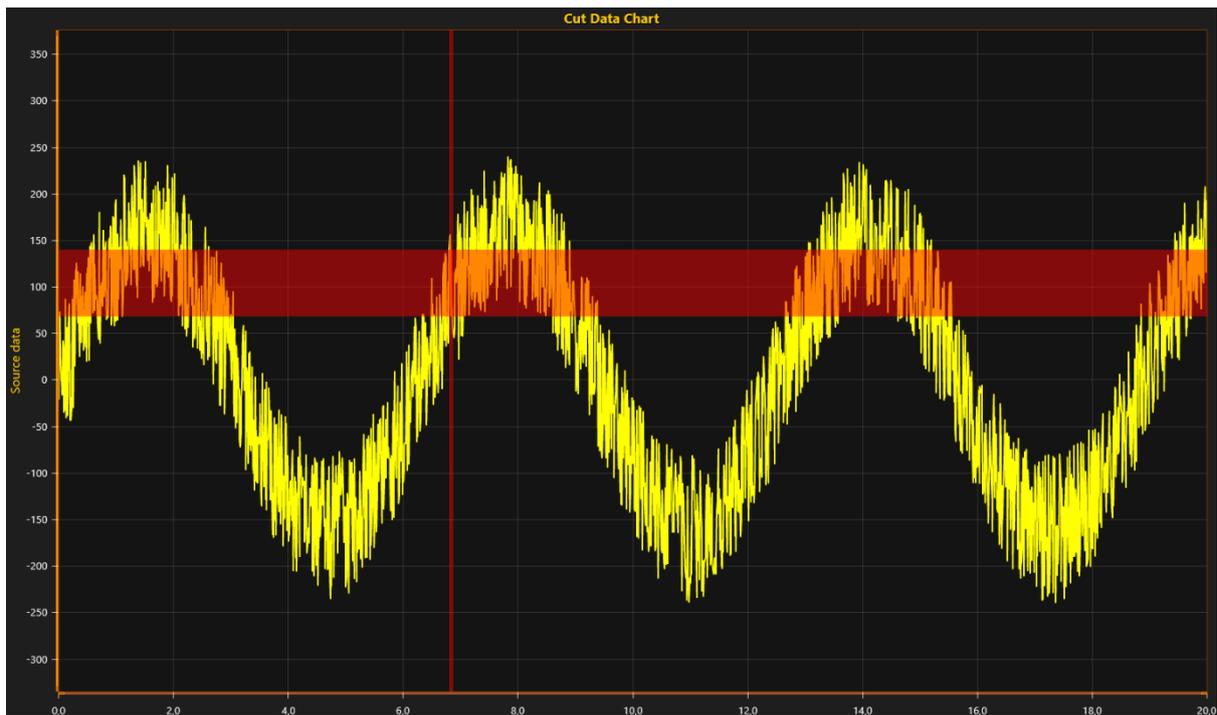


Figure 5-122. Hair-cross cursor with Y range indicator (IndicateTrackingYRange = true).

5.29.1 Solving the data values in the position of LineSeriesCursor

The series implementing *ITrackable* interface can be solved by X screen coordinate or by X axis value.

Trackable series have methods for accurate and coarse value solving. The accurate method **SolveYValueAtXValue** loops through data points if necessary and finds the nearest data point match. The coarse method **SolveYCoordAtXCoord** uses cached rendering data of the series for solving the matching Y screen coordinate.

5.29.1.1 Accurate method, solving Y value by X value using data points array

```

LineSeriesValueSolveResult result =
    series.SolveYValueAtXValue(cursor.ValueAtXAxis);

if (result.SolveStatus == LineSeriesSolveStatus.OK)
{
    //PointLineSeries may have two or more points at same X value. If so, center
    it between min and max
    yValue = (result.YMax + result.YMin) / 2.0;
    return true;
}

```

Note! When *cursor.SnapToPoints* is disabled, the *SolveYValueAtXValue* returns interpolated value between the points adjacent to it (the intersection of cursor line and the series line).

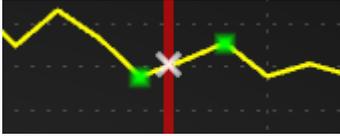


Figure 5-123. *SolveYValueAtXValue* interpolates the value between adjacent data points when *SnapToPoints* is disabled.

5.29.1.2 Coarse method, solving Y screen coordinate by X coordinate using data points array

```
LineSeriesCoordinateSolveResult result =
series.SolveYCoordAtXCoord((int)Math.Round(fCoordX));

if (result.SolveStatus == LineSeriesSolveStatus.OK)
{
    fCoordY = (result.CoordBottom + result.CoordTop) / 2f;
    if (axisY.CoordToValue((int)Math.Round(fCoordY), out yValue) == false)
    {
        return false;
    }
}
```

When the series holds a lot of data points, say > 100.000, it's typically much faster to use the coarse method. The coarse method can be very inaccurate if the chart size or Y segment height in pixels is low.

The coarse method's screen coordinates can be converted into axis values by calling *CoordToValue* method of X and Y axis.

It is a good practice to use an *AnnotationXY* object to display values next to the cursor, see 5.20

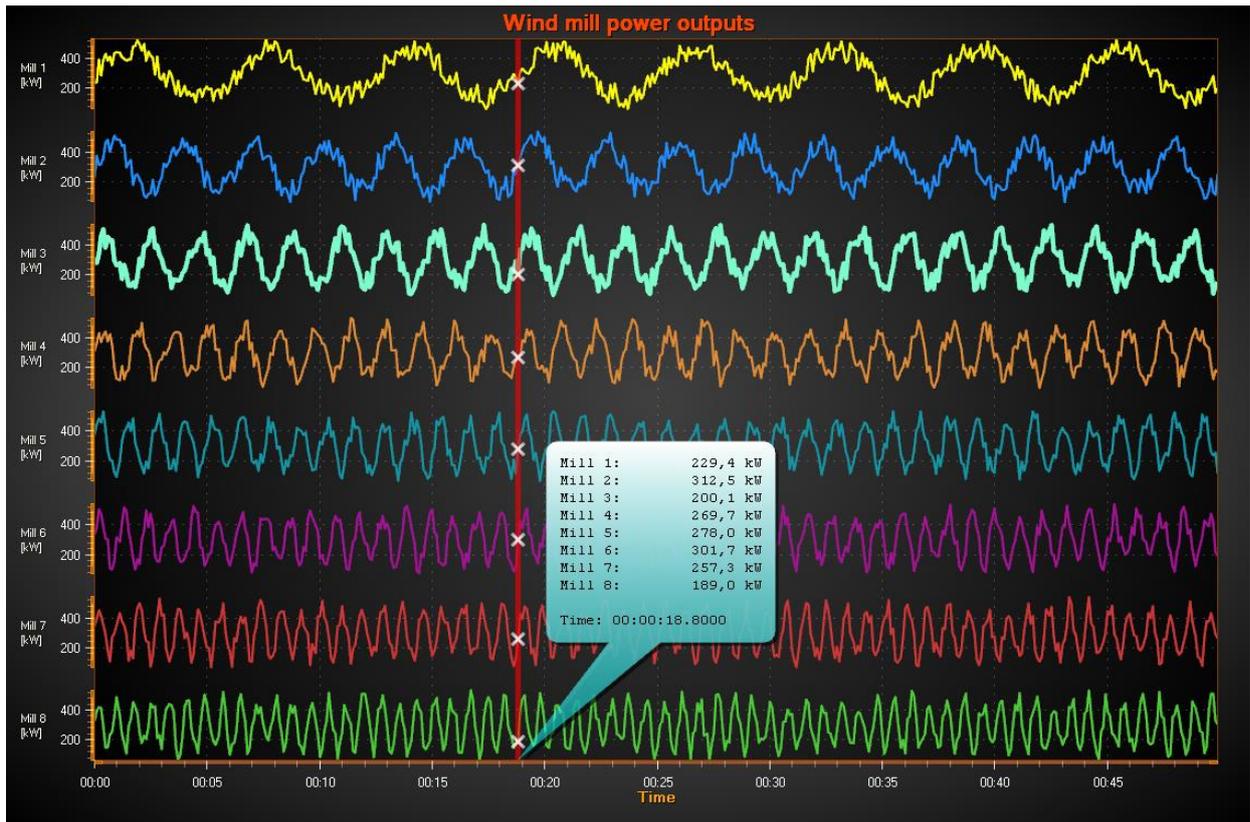


Figure 5-124. LineSeriesCursor used to track PointLineSeries. The values are shown in an AnnotationXY object.

5.30 Event markers

Event markers allow marking a point of interest, where something special occurred during real-time monitoring, or if just wanting to mark a piece of data with a special annotation. Define the marker symbol with **Symbol** property and a text label with **Label** property. Set the vertical position with **VerticalPosition** property and use **Offset** to shift the object property if necessary. All markers must be assigned with **XValue**, which sets the marker's position on X axis.

To select the shape of the marker, set **Symbol.Shape**. Available shapes are **Rectangle**, **Circle**, **Triangle**, **Flag**, **FlagLightning**, **Cross**, **CrossAim**, **Bitmap**, **HollowBasic**, **HollowBasicActive**, **HollowHarmonic**, **HollowActiveSideband**, **HollowSideband**, **HollowTailedActive** and **HollowTailed**.

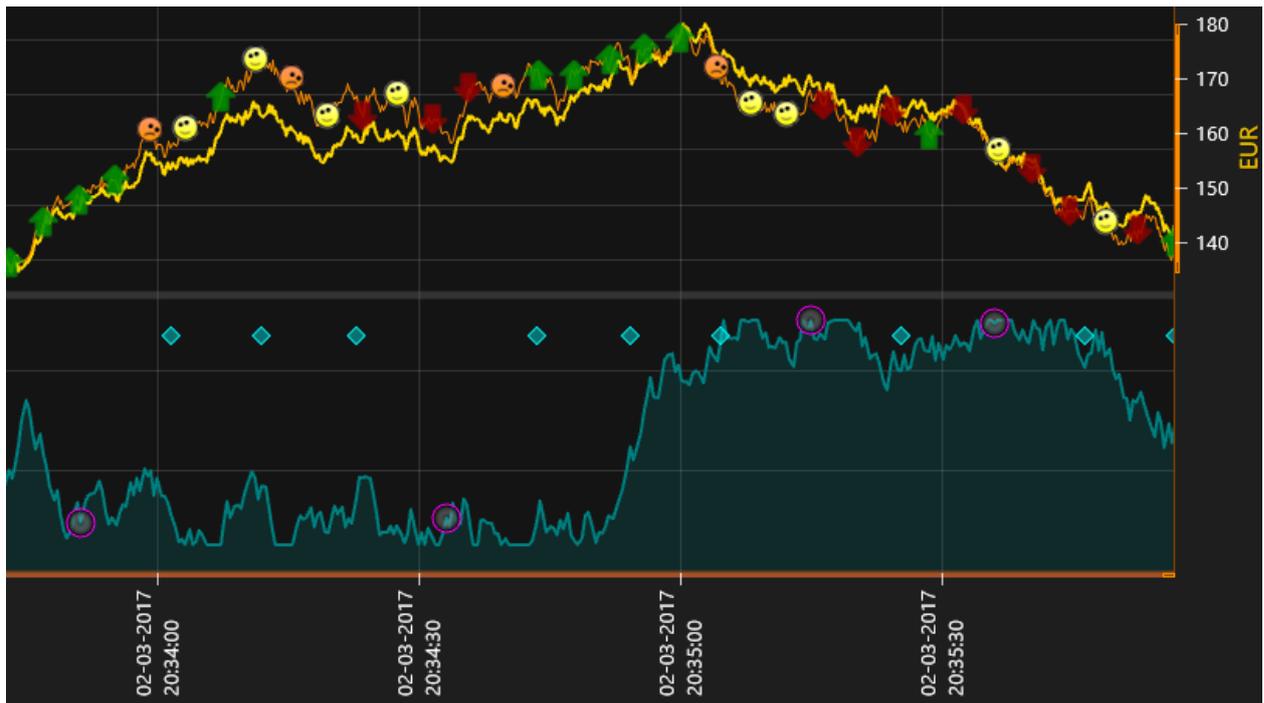


Figure 5-125. Markers in trading example.

5.30.1 Chart event markers

ChartMarkers collection property allows adding chart markers. A chart marker can be used to indicate a point of interests, like “Test person stood up”, “Capacitor bypassed”. Unlike series event markers, chart event markers are not attached to a specific series. The markers can be dragged with mouse into another location.

5.30.2 Line series event markers

Line series have **SeriesEventMarkers** collection property. It can be used to assign series specific event markers. The series event markers can be dragged with mouse to another location, while keeping the event marker attached to series values. To enable this, marker’s **VerticalPosition** must be set to **TrackSeries**. This is available for series implementing **ITrackable** interface.

By setting **HorizontalPosition** to **SnapToPoints**, the marker aligns itself horizontally to position of nearest data point. **HorizontalPosition = AtXValue** allows placing the marker at any x value. Respectively, **VerticalPosition = AtYValue** allows setting the marker vertically to any Y level.

In addition to normal set of shapes, **SeriesEventMarker** supports two special **Symbol.Shape** settings, **HollowYAxis** and **HollowYAxisActive**, which allow vertical line with Y axis ticks projection. They have a

pixel wide vertical line which picks positions of **MajorTicks** and **MinorTicks** from the Y axis the series is attached to. To adjust the tick lengths, edit **YAxis.MajorDivTickStyle.LineLength** and **YAxis.MinorDivTickStyle.LineLength** properties.

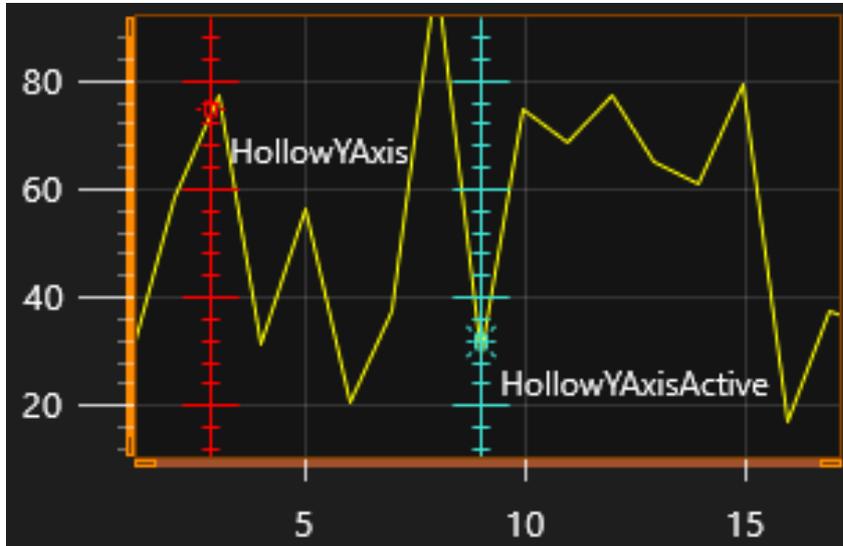


Figure 5-126. Two special SeriesEventMarkers shapes: HollowYAxis and HollowYAxisActive. Very handy when making per-series data cursors.

5.31 Persistent series rendering layers

PersistentSeriesRenderingLayer can be used for extremely fast rendering of repetitive line/points data, or line/points/high-low/area fill data that is plotted in same X and Y range over and over again.

For example, consider a case of FFT monitoring: Every second 20 new data strips are received. The newest data should be visible as well as all the historic traces. But the monitoring lasts for hours. By rendering this kind of data with regular rendering, $20 * 60 * 60 = 72000$ new line series are needed every hour. The PC will run out of memory probably before 1 hour is monitored. It is certain that rendering will slow down so badly that it's not usable anymore.

PersistentSeriesRenderingLayer is kind of a bitmap, that allows adding rendering data incrementally in it. It keeps the graphics until cleared by command. This way, each update round, only one series needs to be rendered on the layer, followed by the layer rendering on the screen. CPU load or memory footprint doesn't rise. If existing data should be faded away gradually, it can be done by multiplying the alpha of the bitmap pixels.

It is possible to create as many **PersistentSeriesRenderingLayer** objects as needed, and any count of series can be rendered on each of them, any update round.

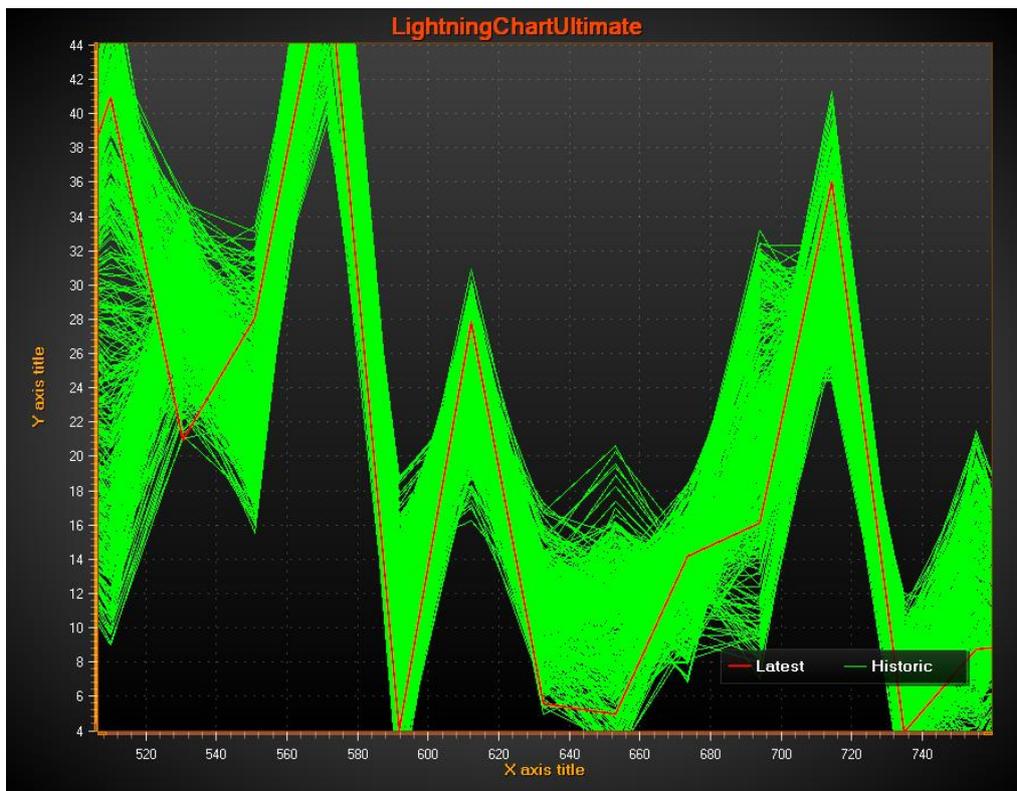


Figure 5-127. Persistent layer shows historical traces, in green color. A regular PointLineSeries is shown over it, in red color.

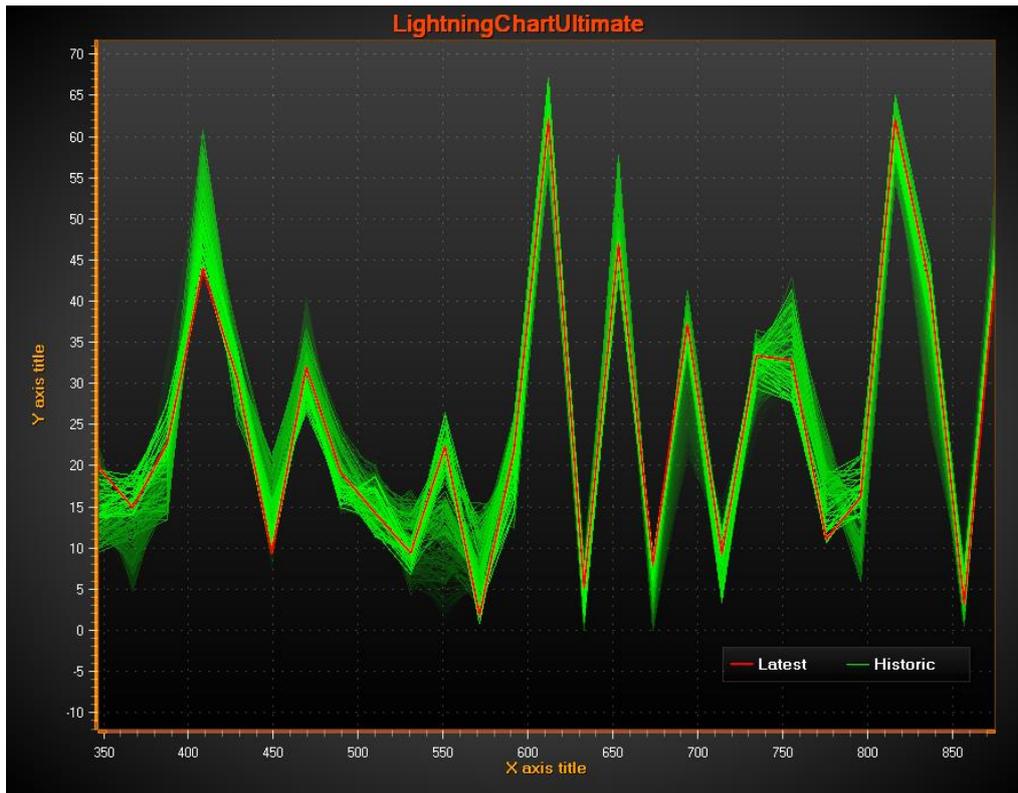


Figure 5-128. Persistent layer shows historical traces, in green color. `MultiplyAlpha` method is called before updating new data in the rendering layer making the oldest traces fade away.

5.31.1 Creating the layer

`PersistentSeriesRenderingLayer` is not a sub-property of `ViewXY` and can't be added with Visual Studio's property grid. **`PersistentSeriesRenderingLayer`** objects must be created in code. Create it as follows:

```
using Arction.[edition].Charting.Views.ViewXY;

PersistentSeriesRenderingLayer layer = new PersistentSeriesRenderingLayer
(m_chart.ViewXY, m_chart.ViewXY.XAxes[0]);
```

By supplying `ViewXY` object as parameter, it binds the layer in `ViewXY`. Supply the same `XAxis` object that is used with the series rendered on it.

Multiple layers will render in the order of creation with the chart.

5.31.2 Clearing the layer

layer.Clear() clears the layer and initializes the color with ARGB=(0,255,255,255).

layer.Clear(Color color) clears the layer with given color. In most cases, it's most useful to set the same color used in the background, but set its A = 0. With black background, use `layer.Clear(Color.FromArgb(0,0,0,0));`

5.31.3 Adjusting layer alpha

MultiplyAlpha(value) allows making the layer more transparent or opaquer. Multiplying effects every pixel in the layer separately.

By supplying value < 1, transparency will be increased (decays the layer).

By supplying value > 1, opacity will be increased (brings the layer more visible).

Takes no effect with value of 1.

For example, **MultiplyAlpha(0.8)** sets the alpha to 80% of existing alpha. **MultiplyAlpha(2)** adjust it to 200%.

5.31.4 Rendering data into the layer

Render the data into the layer by using any of **PointLineSeries**, **SampleDataSeries**, **FreeformPointLineSeries**, **HighLowSeries** or **AreaSeries** objects. They can be series that have been added into **ViewXY.PointLineSeries**, **ViewXY.SampleDataSeries**, **ViewXY.FreeformPointLineSeries**, **ViewXY.HighLowSeries** or **ViewXY.AreaSeries** collection. A temporary series that have not been added into these collections can also be used. Fill the data in the series as usual (see chapter 5.6.4 for **PointLineSeries**, 5.7.2 for **SampleDataSeries**, 5.8 for **FreeformPointLineSeries**, 5.10.4 for **HighLowSeries** and 5.11.1 for **AreaSeries**).

layer.RenderSeries(PointLineSeriesBase series): Render one series on the layer.

layer.RenderSeries(List<PointLineSeriesBase> seriesList): Render all given series on the layer. More efficient than calling **layer.RenderSeries(PointLineSeriesBase series)** for each series separately.

Note! All the given series will be rendered on the layer, even if their **Visible** is set to **False**.

Note! The X axis that used with the series must be the same as the one supplied for **PersistentSeriesRenderingLayer** constructor. Otherwise, the series object will be skipped.

Note! `RenderSeries` is for rendering INTO the layer. The layer itself will be rendered just before regular series (`PointLineSeries`, `SampleDataSeries`, `FreeformPointLineSeries`, `HighLowSeries`, `AreaSeries`).

5.31.5 Disposing the layer

To dispose the layer and prevent it from rendering with the chart, call `layer.Dispose()`.

5.31.6 Anti-aliasing data in the layer

To anti-alias the data in the chart rendering stage, set `layer.AntiAliasing` to `True`. This enables anti-aliasing also if the hardware doesn't support it.

5.31.7 Getting list of layers

`ViewXY.GetPersistentSeriesRenderingLayers()` returns list of all created layers, including `PersistentSeriesRenderingIntensityLayers`.

5.31.8 Some layer limitations to be aware of

Due to its special rendering technique, please keep these limitations in mind:

- X axis `ScrollMode` must be set to `None`. Real-time scrolling of X axis is not possible in this approach.
- Zooming, panning, axis adjustment and chart resize will cause the image to be in un-sync with axis ranges. These features should be disabled when using persistent plotting, or the application logic made so that it clears the layer and recreates temporarily the older line series for new layer rendering (there are event handlers for axis range change and resize).
- Chart resizing will clear the layer, as well as resuming from Windows desktop lock state.
- Mouse interactivity is not supported on the series rendered only on the layer
- EMF/WMF/SVG export, copy to clipboard in vector format, and print in vector format don't support the layer. Only raster formats are supported.

5.32 Persistent series rendering intensity layers

PersistentSeriesRenderingIntensityLayer allows collecting traces into a layer and coloring it by the hit count per pixel. The coloring is made by using a value-range palette. The traces can be used with the same series types as in **PersistentSeriesRenderingLayer** (see chapter 5.31). They are very much similar to it, main difference being the coloring. When rendering a trace in a location of a pixel again with second rendering call, the intensity of it grows, increasing its value in the value-range palette.

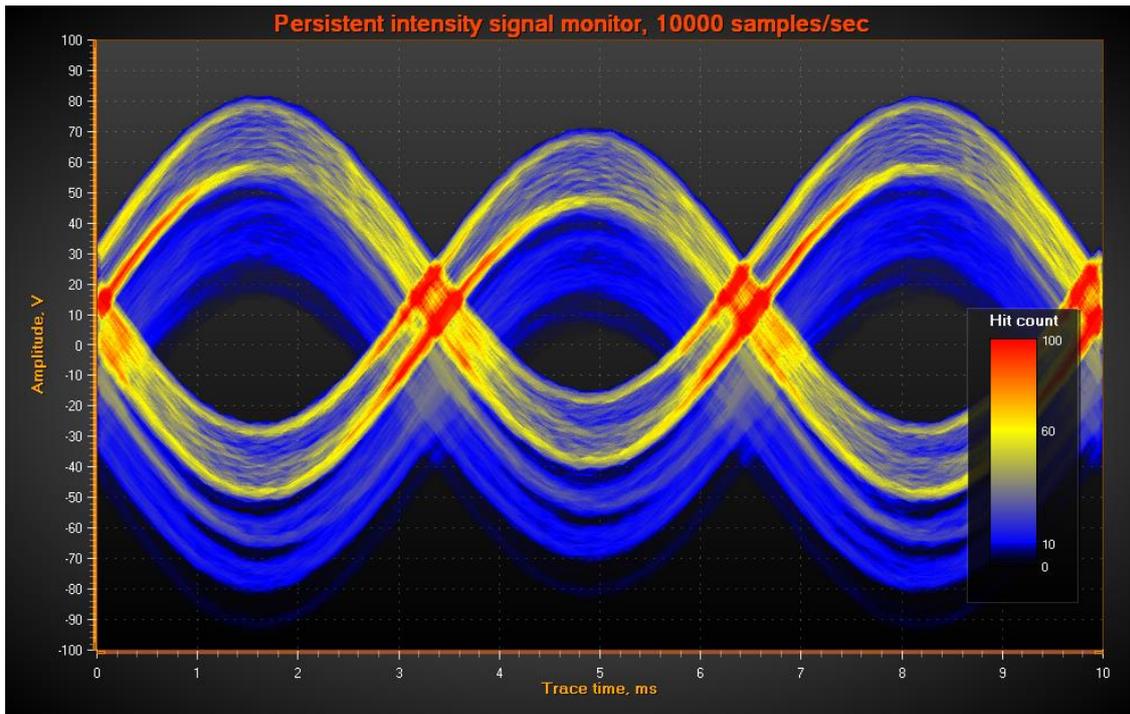


Figure 5-129. Persistent intensity layer highlights areas of concentrated activity, in this case in yellow and red.

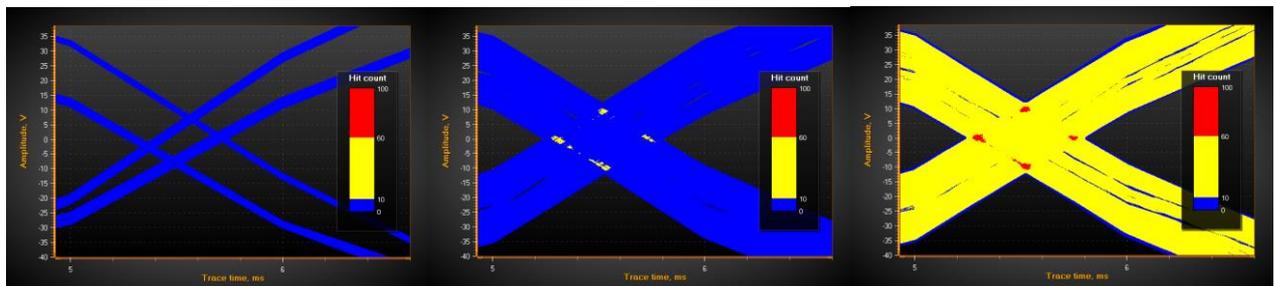


Figure 5-130. Repetitive signal trace is rendered in the same region. On the left, only a couple of traces have been rendered on the layer, showing all colors in blue. On the middle, a lot of traces have been rendered, but mostly on different coordinates. In the intersections of the traces, the hit count exceeds the trace count of 10 defined in the palette for yellow color threshold. In the rightmost image, hundreds of traces have been rendered in total, and intersections start to exceed threshold defined for red color.

5.32.1 Creating the layer

The **PersistentSeriesRenderingIntensityLayer** is not a sub-property of ViewXY and can't be added with Visual Studio's property grid. **PersistentSeriesRenderingIntensityLayer** objects must be created in code.

Create it as follows:

```
using Arction.LightningChartUltimate.Views.ViewXY;

PersistentSeriesRenderingIntensityLayer layer = new
PersistentSeriesRenderingIntensityLayer(m_chart.ViewXY,
m_chart.ViewXY.XAxes[0]);
```

5.32.2 Clearing the layer

layer.Clear() clears the layer and resets the counters.

5.32.3 Changing palette colors

Define the palette type and steps in **ValueRangePalette** property of the layer. **ValueRangePalette.Type = Gradient** makes a gradient coloring, **ValueRangePalette.Type = Uniform** makes the layer render with discrete color steps.

5.32.4 Adjusting the intensity effect of new trace and decay of old traces

Use **NewTraceIntensity** property to control how great intensity effect the new trace rendered with **RenderSeries** call gets. Typical value is 1...100, depending on how fast the color range is set to fill up with the traces.

Use **HistoryIntensityFactor** to adjust the decay speed of the old traces. Typical value is in range of 0.5 – 0.99.

Note that setting **HistoryIntensityFactor** itself doesn't update the layer until the next call of **RenderSeries**.

5.32.5 Rendering data into the layer

Render a **PointLineSeries**, **FreeformPointLineSeries**, **SampleDataSeries**, **HighLowSeries** or **AreaSeries** to the layer by **RenderSeries** method.

layer.RenderSeries(PointLineSeriesBase series): Render one series on the layer.

layer.RenderSeries(List<PointLineSeriesBase> seriesList): Render all given series on the layer. No performance gain over ***layer.RenderSeries(PointLineSeriesBase series)*** though.

When the data is updated into the layer, ***NewTraceIntensity*** is used for the new trace. Old trace data is decayed with ***HistoryIntensityFactor*** at the same time. ***layer.RenderSeries(List<PointLineSeriesBase> seriesList)*** decays old traces after every series object.

5.32.6 Disposing the layer

To dispose the layer and prevent it from rendering with the chart, call ***layer.Dispose()***.

5.32.7 Anti-aliasing data in the layer

To anti-alias the data in the chart rendering stage, set ***layer.AntiAliasing*** to ***True***. It enables the anti-aliasing also if the hardware doesn't support it.

5.32.8 Getting list of layers

ViewXY.GetPersistentSeriesRenderingLayers() returns list of all created layers, including ***PersistentSeriesRenderingLayers***.

6. View3D

View3D allows visualizing data in 3D space. 3D model can be zoomed, rotated and lit up with various ways. **Different series types can be placed into the same 3D view to make a combined visualization.**

View3D	3D chart view
Annotations	(Collection)
AutoSizeMargins	False
BarSeries3D	(Collection)
> BarViewOptions	
> Border	Border
> Camera	
ClipContents	False
> Dimensions	
> FrameBox	FrameBox - col -1
> LegendBox	LegendBox3D
Lights	(Collection)
> Margins	0, 0, 0, 0
MeshModels	(Collection)
> OrientationArrows	
PointLineSeries3D	(Collection)
Polygons	(Collection)
Rectangles	(Collection)
SurfaceGridSeries3D	(Collection)
SurfaceMeshSeries3D	(Collection)
VolumeModels	(Collection)
> WallOnBack	WallXY
> WallOnBottom	WallXZ
> WallOnFront	WallXY
> WallOnLeft	WallYZ
> WallOnRight	WallYZ
> WallOnTop	WallXZ
WaterfallSeries3D	(Collection)
> XAxisPrimary3D	HighlightingItemBase
> XAxisSecondary3D	HighlightingItemBase
> YAxisPrimary3D	HighlightingItemBase
> YAxisSecondary3D	HighlightingItemBase
> ZAxisPrimary3D	HighlightingItemBase
> ZAxisSecondary3D	HighlightingItemBase
> ZoomPanOptions	

Figure 6-1. View3D object main tree.

6.1 3D model and dimensions

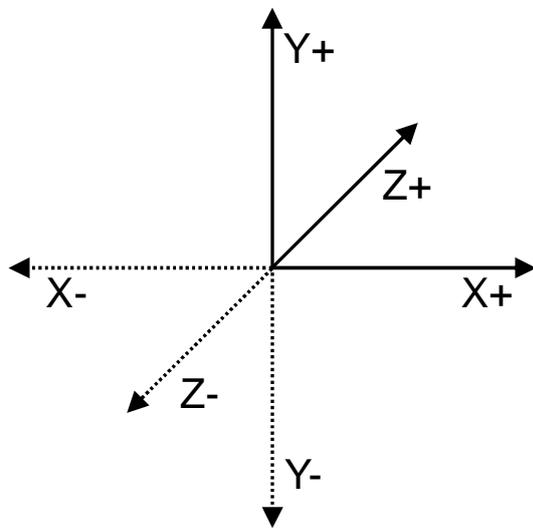


Figure 6-2. 3D model positive and negative directions.

3D model is constructed in the center of 3D world. Dimension magnitudes define the size of the model box in 3D space. Walls and axis sizes are defined with this dimension box. Use the **Dimensions** property to set each dimension magnitude.

When camera rotation is not defined, positive X direction is to the right, positive Y dimension upwards and positive Z direction inwards to the screen.

6.1.1 World coordinates

Some 3D objects use “*World coordinates*”, not axis values. For example, lights are positioned this way to be independent from axis ranges. World coordinates can be called also as “*3D model space coordinates*”.

The origin [0,0,0] is in the center of the model. The actual 3D model space ranges from [-Dimensions.X/2 to Dimensions.X/2], [-Dimensions.Y/2 to Dimensions.Y/2] and [-Dimensions.Z/2 to Dimensions.Z/2].

LightningChart provides methods to convert values between series values, axis values, world coordinates and screen coordinates. See the demo application examples and help documentation for details.

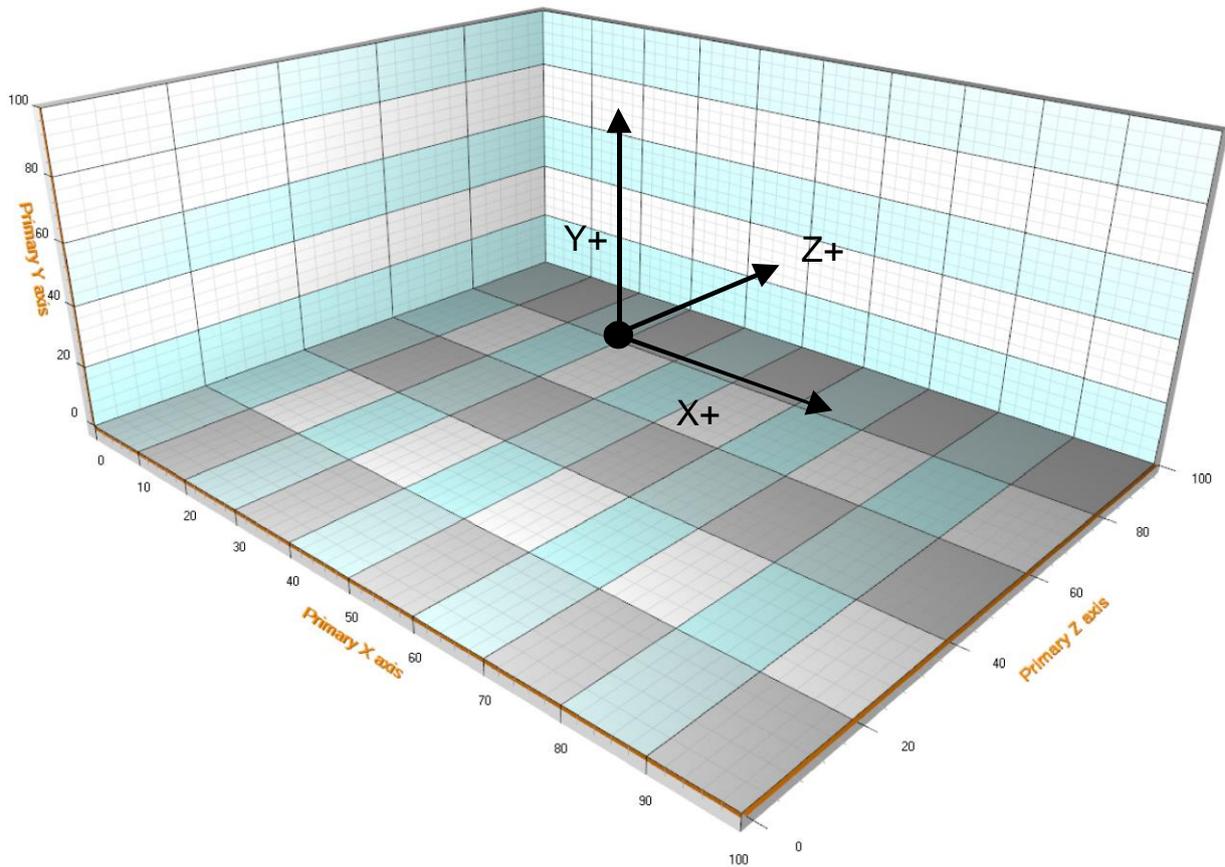


Figure 6-3. Example of 3D view setup. Dimensions are set to X=100, Y=40, Z=80. Walls visible: left, bottom, back. Perspective camera is used.

6.2 Walls

Walls (*WallOnFront*, *WallOnBack*, *WallOnTop*, *WallOnBottom*, *WallOnLeft*, *WallOnRight*) are used to present axis grids and gridstrips and to give a base for the axes. By default, bottom, left, right, back and front walls are visible. Their **AutoHide** property is set true. When rotating the view, the obstructing walls are temporarily hidden so that they don't block the view of chart contents. To force a wall visible, set **Visible = true** and **AutoHide = false**.

Use *XGridAxis*, *YGridAxis*, *ZGridAxis*, *GridStripColorX*, *GridStripColorY*, *GridStripColorZ* and **GridStrips** properties to select from which axes the grid is applied, and to modify the coloring of the grid strips. The available properties depend on the wall orientation.

6.3 FrameBox

A simplified 3D box presentation can be used instead of walls. Set `Visible = false` for every wall, then set `FrameBox.Style = AllEdges`. Set the color or the frame with `FrameBox.LineColor`.

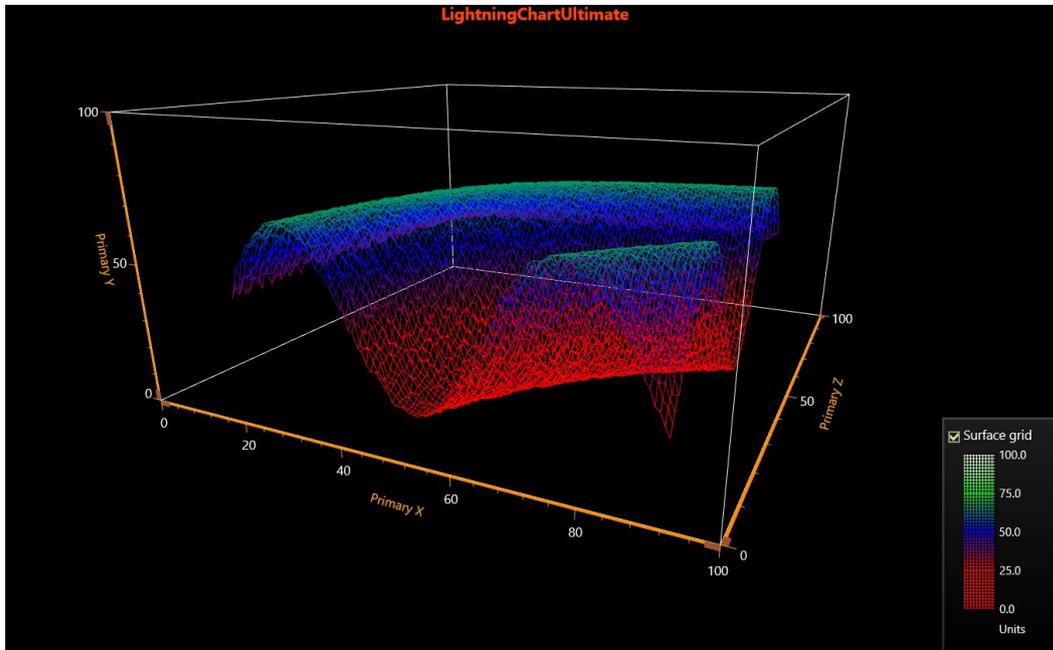


Figure 6-4. FrameBox visible, walls are hidden.

6.4 Camera

Camera	
FieldOfViewAngle	45
MinimumViewDistance	50
OrientationMode	ZXY_Extrinsic
OrthographicCamera	False
Projection	Perspective
RotationX	20
RotationXMaximum	720
RotationXMinimum	-720
RotationY	-25
RotationYMaximum	720
RotationYMinimum	-720
RotationZ	0
RotationZMaximum	720
RotationZMinimum	-720
> Target	
ViewDistance	180

Figure 6-5. Camera properties.

Camera type, location, distance and target together determine the 3D viewpoint. Use **RotationX**, **RotationY**, **RotationZ** and **ViewDistance** to set the camera position in the 3D model space. Target the camera to preferred direction by setting the **Target** property.

Select projection type with **Projection** property.

- **Perspective**, shows a realistic projection.
- **Orthographic**, projection type used in scientific and engineering applications. This selection is recommended over **OrthographicLegacy**.
- **OrthographicLegacy**, (equivalent to OrthoGraphicCamera = True in LightningChart v.8.3 and earlier). This is slower to render after zooming compared to Orthographic. It maintains the sizes of the 3D objects, if they are defined in 3D world coordinates (not axis values). Also, the thickness of the walls stays the same when zooming. Zooming changes the dimensions but does not affect **ViewDistance**.

RotationX, **RotationY** and **RotationZ** can be limited by setting boundaries via **RotationXMinimum**, **RotationXMaximum**, **RotationYMinimum**, **RotationYMaximum**, **RotationZMinimum** and **RotationZMaximum** properties.

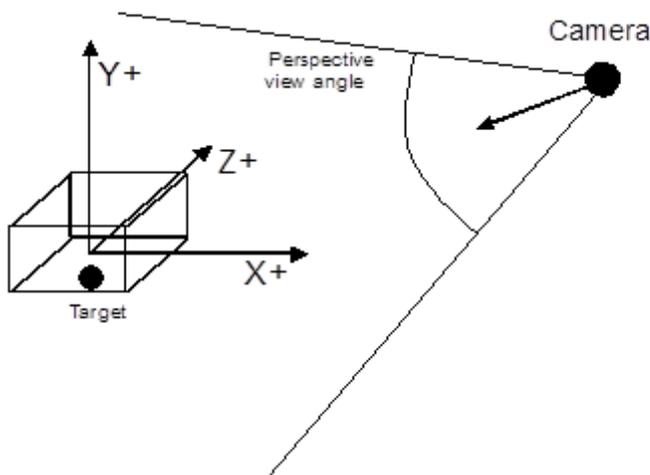


Figure 6-6. Perspective camera presentation in 3D space.

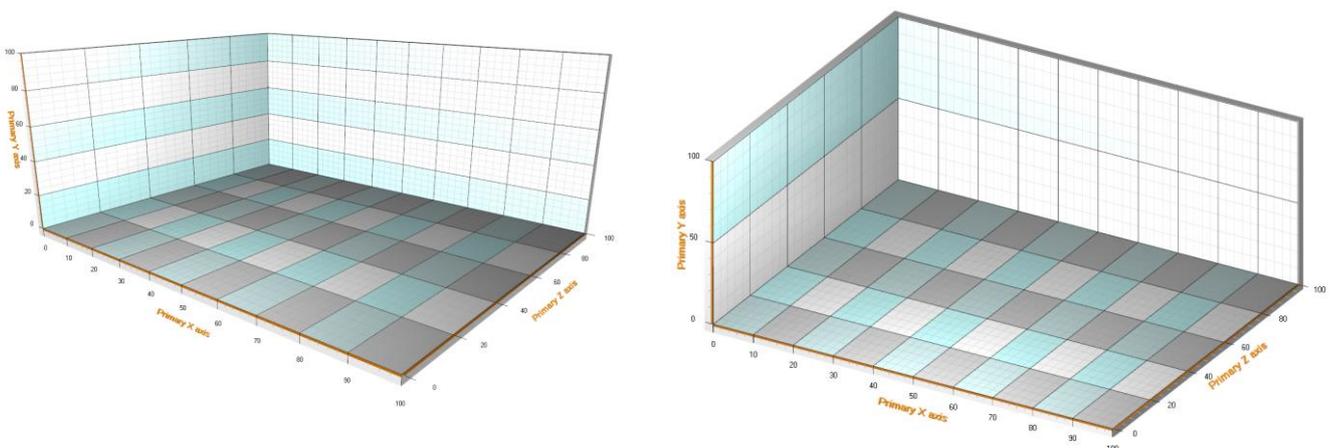


Figure 6-7. Perspective and orthographic camera views in 3D space.

Zoomed view in Orthographic and OrthographicLegacy differ as follows:

Orthographic

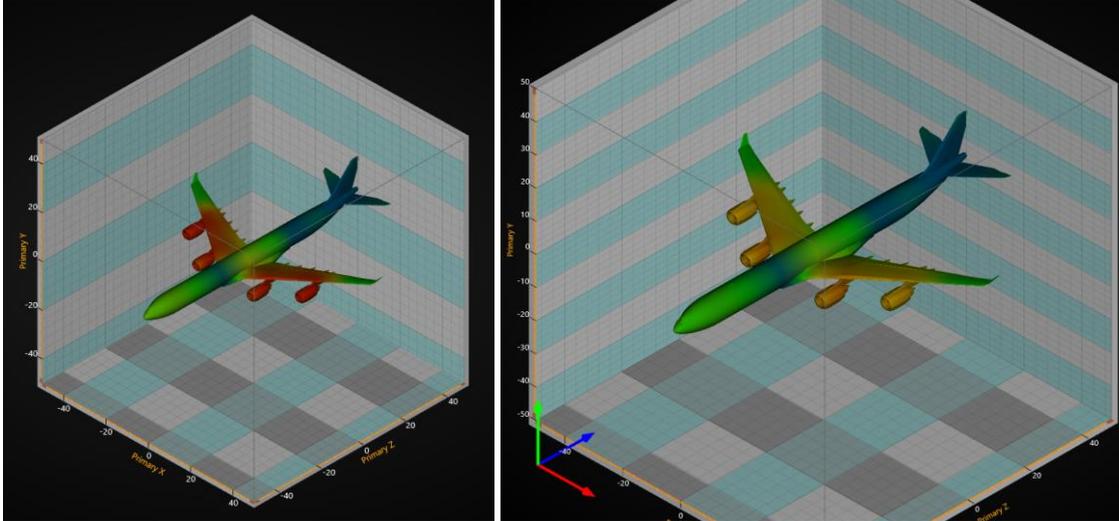


Figure 6-8. Orthographic projection type. On the left, unzoomed. On the right, zoomed in. Airplane (MeshModel3D) object size grows on the screen along with other objects.

OrthographicLegacy

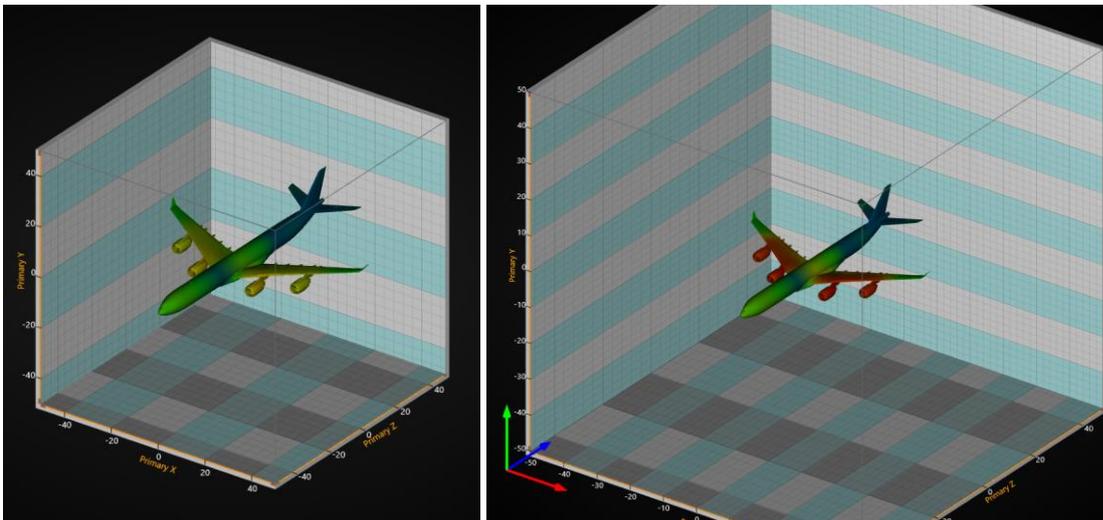


Figure 6-9. OrthographicLegacy. On the left, unzoomed. On the right, zoomed in. Airplane (MeshModel3D, see 6.14) object size stays the same but 3D dimensions are changed.

6.4.1 Predefined cameras

Use **SetPredefinedCamera** method of **View3D.Camera** to set one of the predefined cameras.

```
// Setting predefined camera orientation
chart.View3D.Camera.SetPredefinedCamera(PredefinedCamera.BackOrthographic);
```

6.4.2 Camera orientation mode

LightningChart v8.4 added a new camera orientation mode with improved camera orientation definition. The new mode called **ZXY_Extrinsic** (the name defines in which order the dimension are calculated) is now set to be the default orientation mode. It fixes many rotation-based issues especially near the poles of the chart (i.e. camera on top of the chart). The old orientation mode, **XYZ_Mixed**, is still available but will most likely become deprecated at some point in the future. Orientations can be accessed via **View3D.Camera.OrientationMode**.

Rotations are also modified by this change. With the new camera orientation mode, one of the axis directions (world unit vectors) is used as the horizontal mouse rotation axis. This is the axis of which the camera is rotated around. Axis determination is automatically done when **RotationX**, **RotationY** or **RotationZ** properties are changed. Closest axis to the camera's up direction is selected as the rotation axis, so that the rotations feel as natural as possible on all occasions.

The new orientation and rotation model allow views to the 3D scene that were previously impossible.

6.5 Lights

Lights can be freely positioned anywhere in the 3D model space. Several lights can be added into **Lights** collection property. There are two different light types: **Directional** and **PointOfLight**.

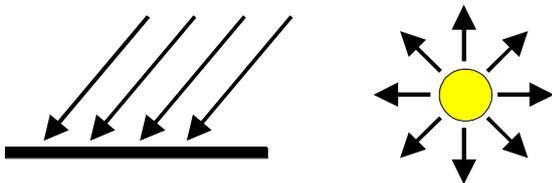


Figure 6-10. Directional light and point of light.

Note! Some series types allow suppressing lighting totally from its surface via **SuppressLighting** property. Check it's not enabled if the series should be correctly lit. Surface series have **LightedSurface** property, which selects the surface side that is correctly lit.

Note! Placing all the lights inside the 3D model box can make the wall edges appear very dark, possibly making axis ticks hardly visible. Adjust axis tick coloring in such case.

6.5.1 Directional light

In **Directional** light, the light rays are parallel, and the light intensity does not attenuate as the distance increases. The light flux gets direction from **Location** and **Target** properties. **LocationFromCamera** property allows using to the location of the camera as a source of light.

6.5.2 Point of light

In **PointOfLight** intensity attenuates as the distance grows. Use **AttenuationConstant**, **AttenuationLinear** and **AttenuationQuadratic** properties to control the attenuation over distance. **Target** is irrelevant with this light type, as the light is distributed equally to all directions.

6.5.3 Lights and materials

All 3D objects have a **Material** property. Material tells how to react to lights. Material's **DiffuseColor** reacts with **DiffuseColor** of a light. Material's **SpecularColor** reacts with light's **SpecularColor**. Diffuse color can be understood as a matte base color, while specular color is the color that reflects off the lit surface. Using high **SpecularPower** gives the object a metallic look.

Surface series have **ColorSaturation** property, valid range is 0...100%. High value boosts the surface fill colors and reduces the shading effect.

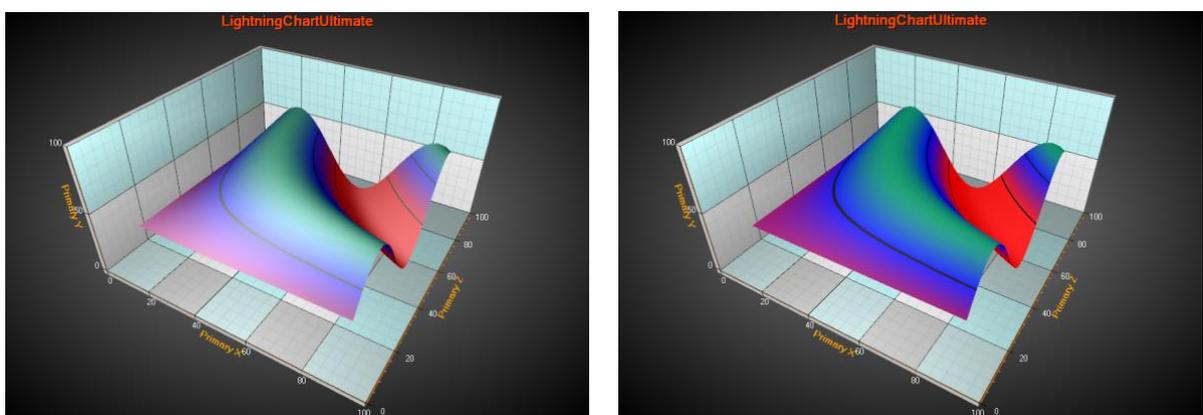


Figure 6-11. The surface series on the left has ColorSaturation = 50%. On the right, ColorSaturation = 85%.

6.5.4 Predefined lighting schemes

Use *SetPredefinedLightingScheme* method of View3D to select a built-in predefined lighting scheme.

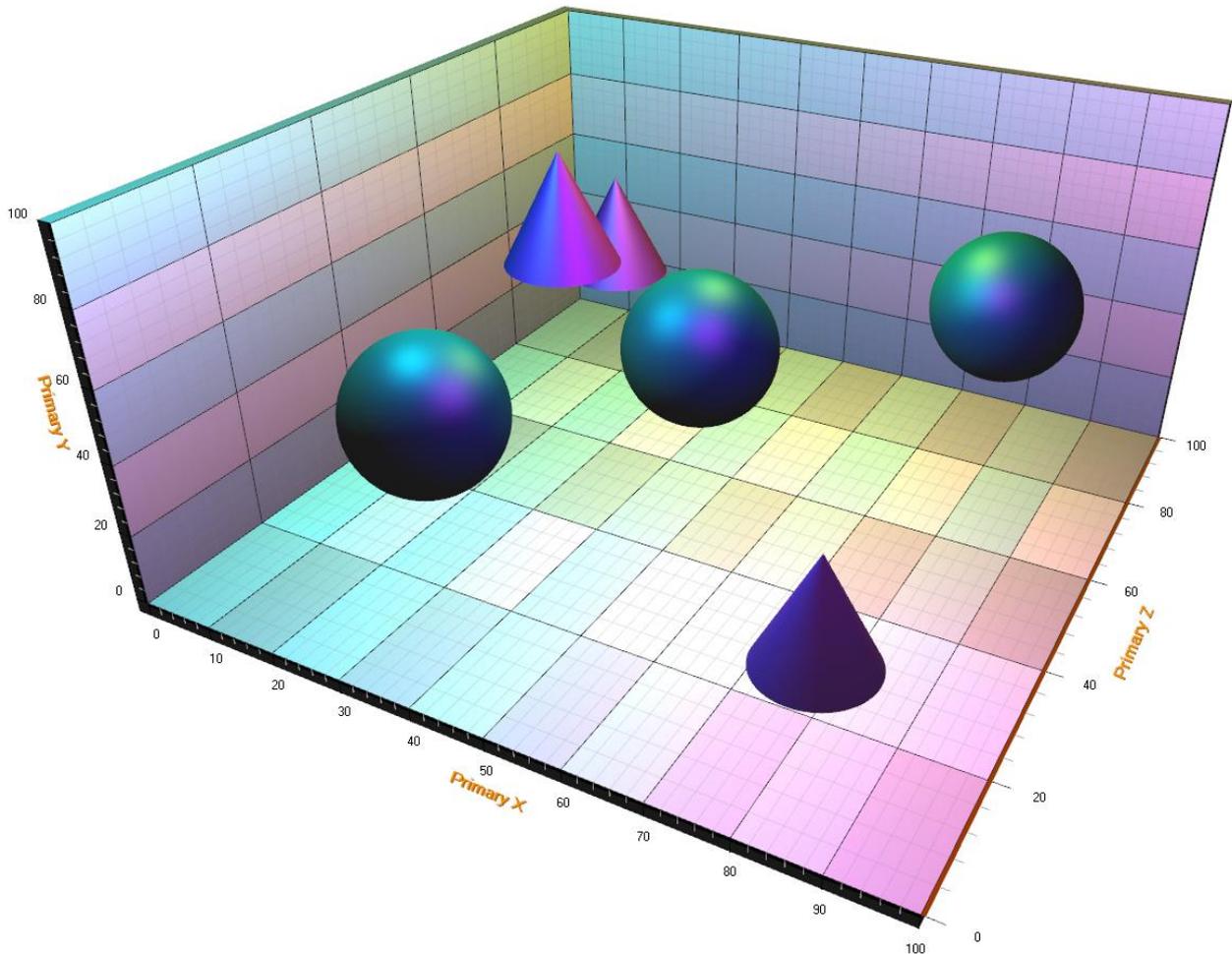


Figure 6-12. Predefined 'DiscoCMY' scheme in use. The scheme is composed from three differently colored PointOfLights near the ceiling. The spheres and cones are made with PointLineSeries3D.

6.6 Axes

For each dimension, there are two axes: primary and secondary. In other words, View3D has the following axis properties available: *XAxisPrimary3D*, *XAxisSecondary3D*, *YAxisPrimary3D*, *YAxisSecondary3D*, *ZAxisPrimary3D* and *ZAxisSecondary3D*.

In general, the 3D axes behave very much like ViewXY's axes. Many of the properties and methods are similar.

6.6.1 Location

The axes can be positioned in 3D model box corners. Use **Location** property of an axis to adjust the position.

- For X axis, the **Location** options are: **BottomFront**, **BottomBack**, **TopFront** and **TopBack**.
- For Y axis, the **Location** options are: **FrontLeft**, **FrontRight**, **BackLeft** and **BackRight**.
- For Z axis, the **Location** options are: **BottomLeft**, **BottomRight**, **TopLeft** and **TopRight**.

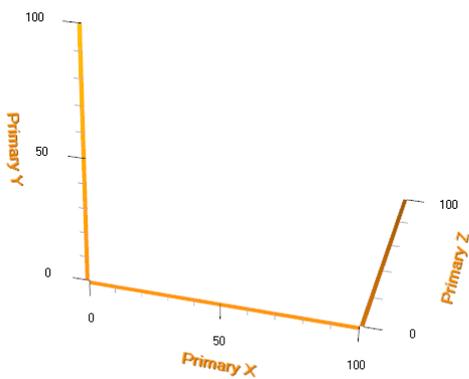


Figure 6-13. Default axis location setup, XAxisPrimary at BottomFront, YAxisPrimary at FrontLeft and ZAxisPrimary in BottomRight.

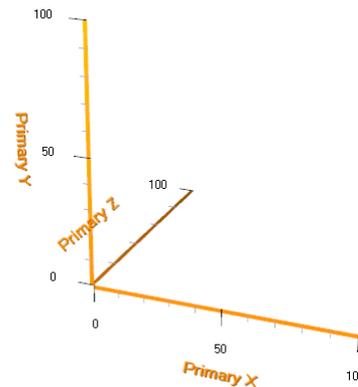


Figure 6-14. ZAxisPrimary location set to BottomLeft.

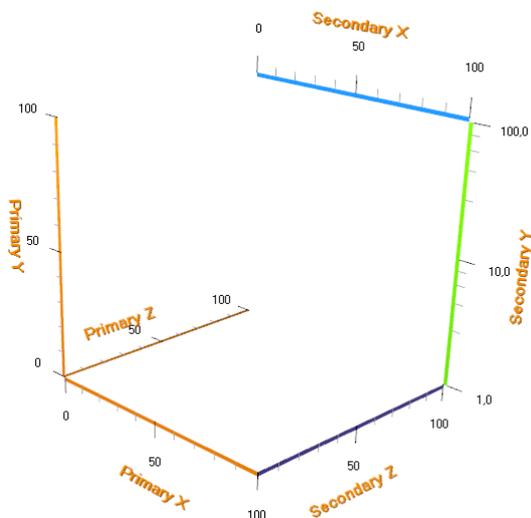


Figure 6-15. Secondary axes set visible and their locations and colors set arbitrarily. Secondary Y axis ScaleType set to Logarithmic.

6.6.2 Orientation

Each axis can be oriented in two planes. This affects the position and orientation of both axis ticks and value labels.

- X axis: XY and XZ planes
- Y axis: XY and YZ planes
- Z axis: XZ and YZ planes

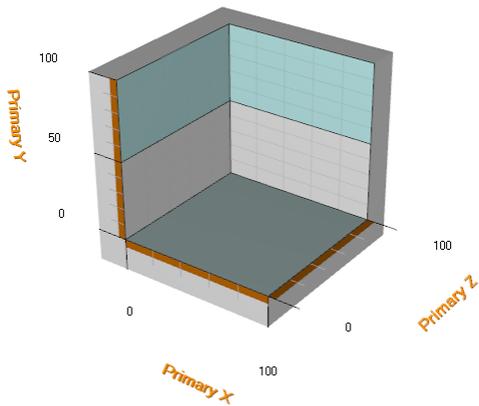


Figure 6-16. X axis orientation is set to XY, Y axis orientation to XY, Z axis orientation to XZ.

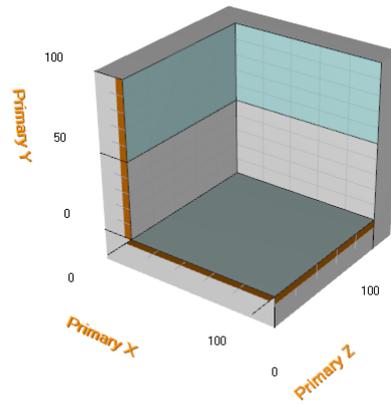


Figure 6-17. Y axis orientation stays same, but X axis orientation is changed to XZ and Z axis orientation is changed to YZ plane.

6.6.3 CornerAlignment

The axis alignment in 3D model box corners can be changed with **CornerAlignment** property. Use **MajorDivTickStyle** and **MinorDivTickStyle Alignment** properties to control the text alignment.

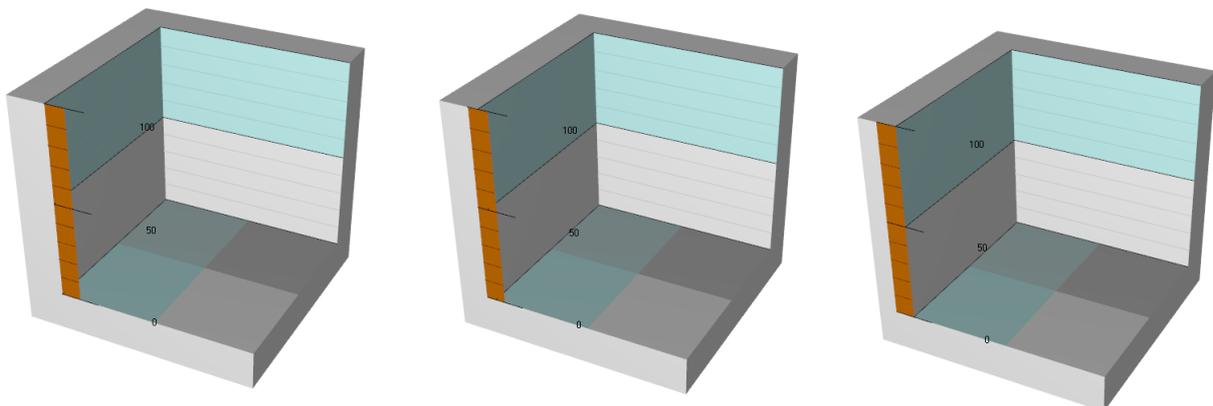


Figure 6-18. Only Y axis is visible in this example. First figure: Y Axis **CornerAlignment** is set to Inside. Alignment properties in **MajorDivTickStyle** and **MinorDivTickStyle** are set to Near. Second figure: **CornerAlignment** is set to AtCorner. Third picture: **CornerAlignment** is set to Outside.

6.7 Margins

From LightningChart v.8.4 onwards, View3D supports margins. Similarly to ViewXY, when **AutoAdjustMargins** is set **true**, the graph size is adjusted so that there's enough space for all the axes and chart title. If it is **disabled**, **View3D.Margins** property applies allowing setting margins manually. By default, **AutoAdjustMargins** is set **false**.

View3D.MarginsChanged event can be set to trigger when a margin has been changed because of for example resizing it.

The contents of the view are automatically clipped outside the margins. All contents are clipped other than the chart title, annotations and legend boxes as their position is defined in screen coordinates, allowing them to be freely positioned on the margins as well. A one-pixel wide border rectangle, **Border**, can be drawn to display where the margins are. By default, the border is not visible in View3D. The color of the rectangle can be changed via **Border.Color**.

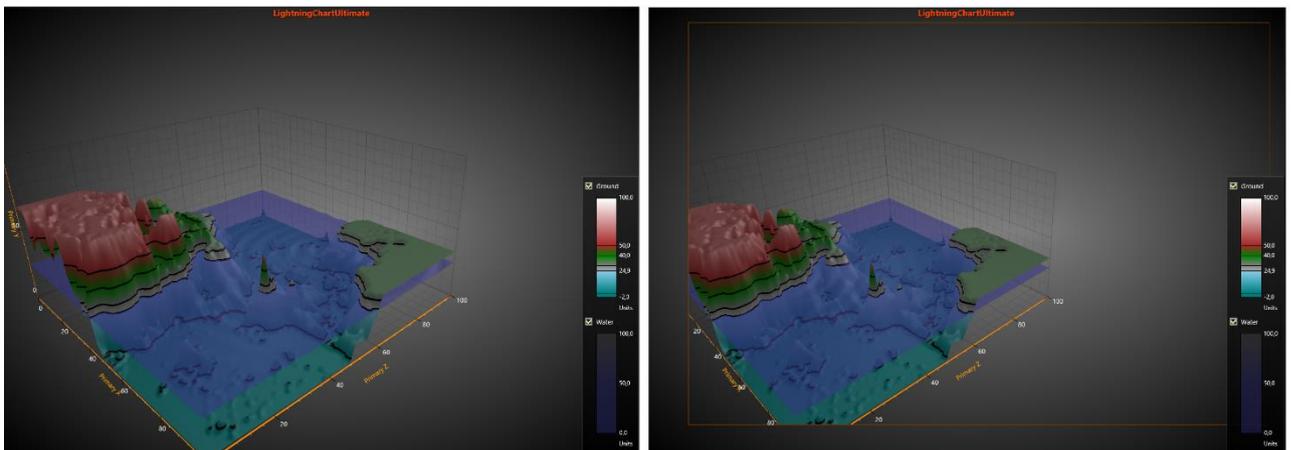


Figure 6-19. On the left, the graph has no margins (all margins set to 0). On the right, margins are set and the content is clipped outside them. **Border.Visible** is set True to mark where the margins of the view are.

6.8 3D series, general

View3D's series allow data visualization in different ways and formats. All series are bound to axis value ranges. For each dimension, the series can be selected to bind to primary or secondary axis. Use **XAxisBinding**, **YAxisBinding** and **ZAxisBinding** properties to control that.

6.9 PointLineSeries3D

PointLineSeries3D allows presenting points and line in 3D space. For points, there are many basic 3D shapes available. Points are connected together with a line, if **LineVisible** property is set **true**.

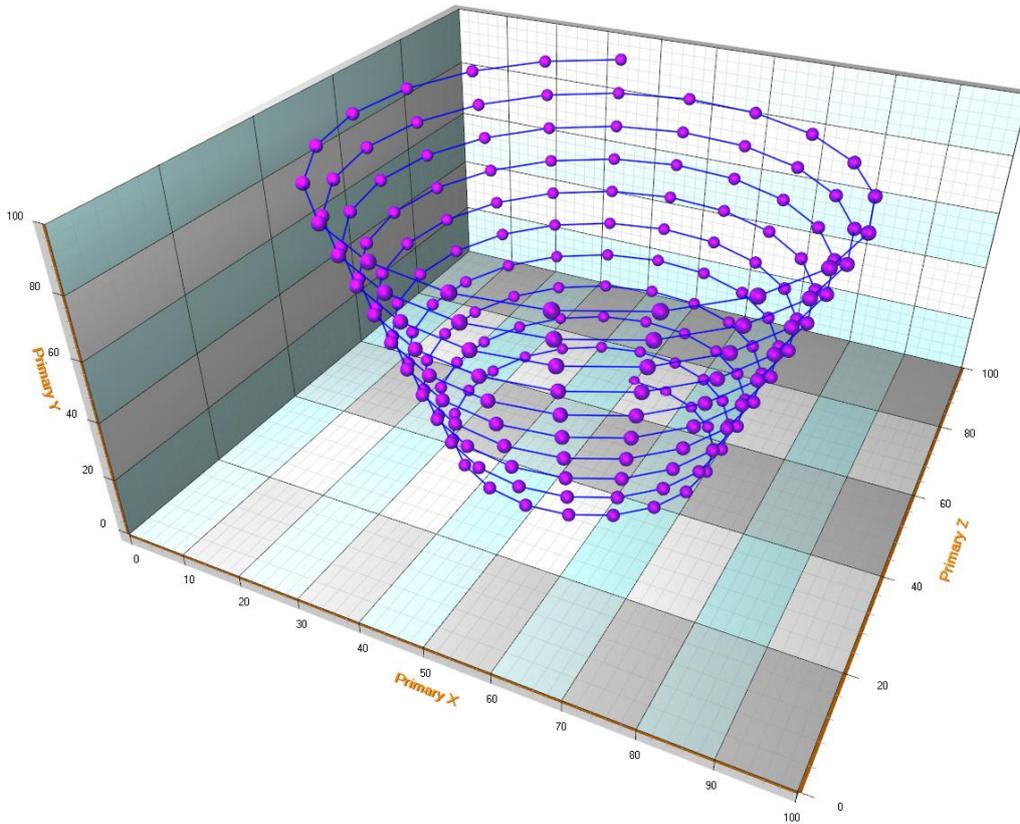


Figure 6-20. A PointLineSeries3D example. PointStyle's Shape is set to Sphere.

6.9.1 Point styles

Points can be shown as real 3D points, or as 2D shapes.

PointStyle	
DetailLevel	30
▶ Rotation3D	
▲ Shape2D	
Angle	45
Antialiasing	True
BitmapAlphaLevel	255
BitmapImage	(none)
BitmapImage TintColor	White
BodyThickness	3
BorderColor	128, 0, 0, 0
BorderWidth	0
Color1	Red
Color2	Black
Color3	Black
GradientFill	Edge
Height	13
LinearGradientDirection	Down
Shape	Cross
UseImageSize	True
Width	13
Shape3D	Box
ShapeType	Shape2D
▶ Size3D	

Figure 6-21. PointStyle property tree. ShapeType can be used to switch between 2D and 3D shapes.

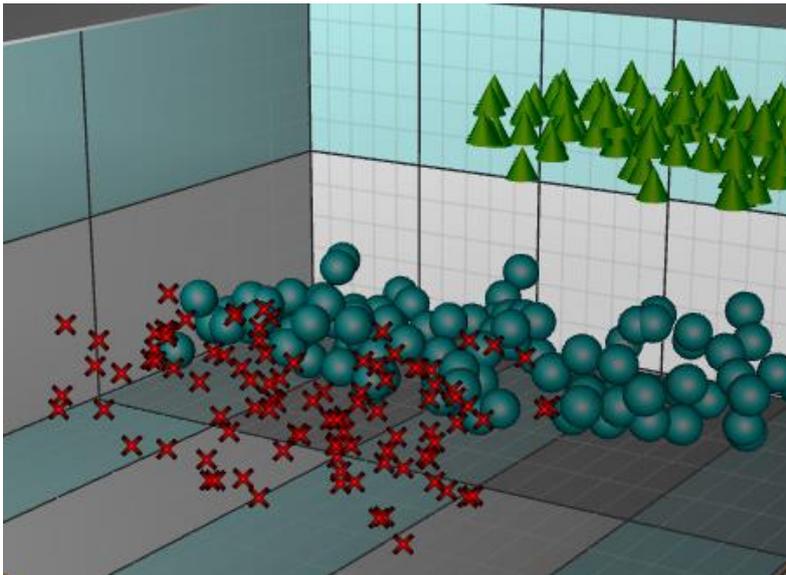


Figure 6-22. Red crosses have ShapeType = Shape2D. Teal and Green objects have ShapeType = Shape3D.

Note! 2D shapes are rendered on top of all 3D objects and they don't have any support for hiding them based on other objects visibility.

6.9.2 Line styles

LineStyle	
AntiAliasing	Normal
Color	Yellow
LineOptimization	Hairline
Pattern	Solid
PatternScale	1
Width	0.2

Figure 6-23. LineStyle properties.

The lines can be rendered as shaded 3D lines or as a one pixel wide hair line.

When having a lot of data in the series, setting **LineOptimization = Hairline** is recommended to avoid performance issues.

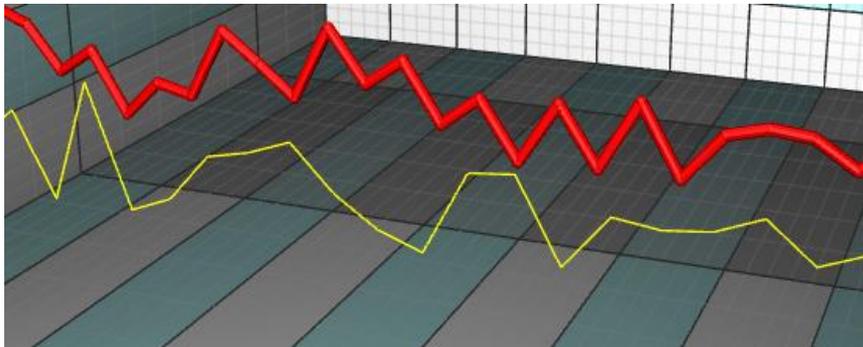


Figure 6-24. Yellow line: LineStyle.LineOptimization = Hairline. Red Line: LineStyle.LineOptimization = NormalShaded.

6.9.3 Adding points

PointLineSeries3D supports three different point formats:

- Points property (SeriesPoint3D array)
- PointsCompact property (SeriesPointCompact3D array)
- PointsCompactColored property (SeriesPointCompactColored3D array)

PointsCompact and **PointsCompactColored** structures are very memory efficient, allowing up to 100 million data points visualization with simple point styles.

Set the point format via **PointsType** property.

6.9.3.1 Points

By using **Points** property, all the advanced coloring options of points are supported. **SeriesPoint3D** structure consists of the following fields:

double X:	X axis value
double Y:	Y axis value
double Z:	Z axis value
Color color:	individual data point color, only applies when IndividualPointColors is enabled, or MultiColorLine is enabled.
float sizeFactor:	size factor multiplies the size defined by PointStyle.Size . Only applies when using IndividualPointSizes is enabled.
object Tag:	freely assignable auxiliary object, for example to attach some details.

Series points must be added in code. Use **AddPoints(...)** method to add points to the end of existing points.

```
SeriesPoint3D[] pointsArray = new SeriesPoint3D [3];
pointsArray [0] = new SeriesPoint3D (50, 50, 50);
pointsArray [1] = new SeriesPoint3D (30, 50, 20);
pointsArray [2] = new SeriesPoint3D (80, 50, 80);

chart.View3D.PointLineSeries3D[0].AddPoints(pointsArray); //Add points to the
end
```

To set whole series data at once while overwriting old points, assign the new point array directly:

```
chart.View3D.PointLineSeries[0].Points = pointsArray; //Assign the points
array
```

6.9.3.2 PointsCompact

PointsCompact property enables low memory consumption, which is important when having a lot of data points.

SeriesPointCompact3D structure consists of the following fields:

float X:	X axis value
float Y:	Y axis value
float Z:	Z axis value

```
SeriesPointCompact3D[] pointsArray = new SeriesPointCompact3D[3];
pointsArray [0] = new SeriesPointCompact3D(50, 50, 50);
pointsArray [1] = new SeriesPointCompact3D(30, 50, 20);
pointsArray [2] = new SeriesPointCompact3D(80, 50, 80);
```

```
chart.View3D.PointLineSeries3D[0].AddPoints(pointsArray); //Add points to the
end
```

To set whole series data at once while overwriting old points, assign the new point array directly:

```
chart.View3D.PointLineSeries[0].PointsCompact = pointsArray; //Assign the
points array
```

6.9.3.3 PointsCompactColored

PointsCompactColored property enables low memory consumption, important when having a lot of data points, but still allows coloring the points with individual colors.

SeriesPointCompactColoured3D structure consists of the following fields:

float X: X axis value

float Y: Y axis value

float Z: Z axis value

int Color: Color of the point

```
SeriesPointCompactColored3D[] pointsArray = new
SeriesPointCompactColored3D[3];
pointsArray [0] = new SeriesPointCompactColored3D(50, 50, 50,
Color.Blue.ToArgb());
pointsArray [1] = new SeriesPointCompactColored3D(30, 50, 20,
Color.Red.ToArgb());
pointsArray [2] = new SeriesPointCompactColored3D(80, 50, 80,
Color.Green.ToArgb());

chart.View3D.PointLineSeries3D[0].AddPoints(pointsArray); //Add points to the
end
```

To set whole series data at once while overwriting old points, assign the new point array directly:

```
chart.View3D.PointLineSeries[0].PointsCompactColored = pointsArray; //Assign
the points array
```

6.9.4 Coloring points individually

By setting **IndividualPointColors = True**, the color fields of points apply instead of **Material.DiffuseColor**.

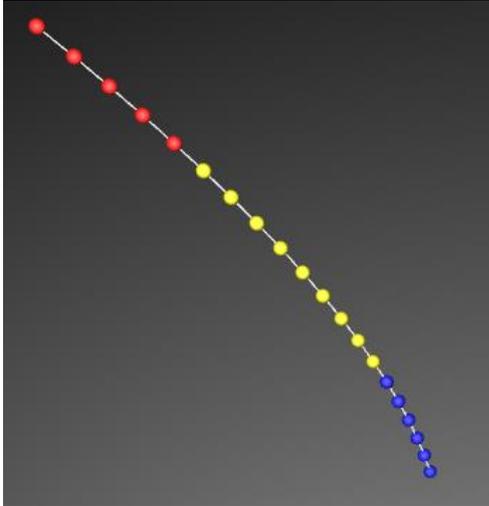


Figure 6-25. *IndividualPointColors* in use.

Note! Individual point coloring is not supported when having *PointsType = PointsCompact*.

6.9.5 Setting points sizes individually

By setting *IndividualPointSizes = True*, *sizeFactor* fields from the points take effect. The factor multiplies the size defined in *PointStyle.Size*.

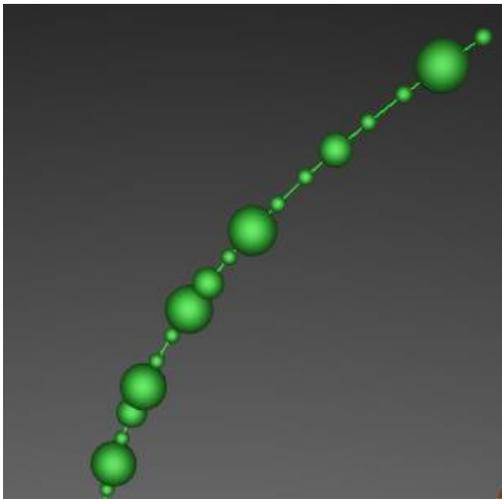


Figure 6-26. *IndividualPointSizes* in use.

Note! Individual point sizes are not supported when having *PointsType = PointsCompact*, or *PointsCompactColored*.

6.9.6 Multi-coloring line

To color the line with given data point colors, set **MultiColorLine = True**. The chart interpolates the color gradients between adjacent points.

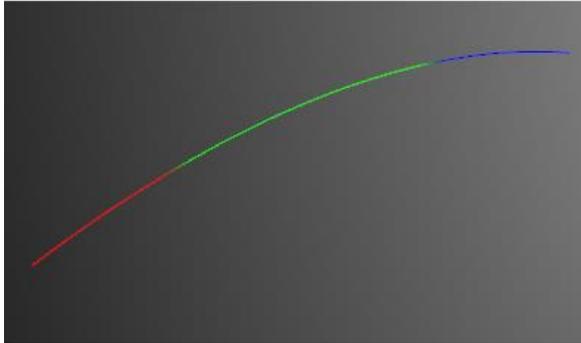


Figure 6-27. MultiColorLine enabled.

Note! MultiColorLine is not supported when having PointsType = PointsCompact.

6.9.7 Displaying millions of scatter points

To be able to show a very high count of scatter points, set **PointsOptimization = Pixels**. Then each series point will be rendered as a single pixel. When having to show 10 million or 100 million data points, use the **PointsCompact** (see 6.9.3.2) or **PointsCompactColored** (see 6.9.3.3) approach to keep memory requirements functional.

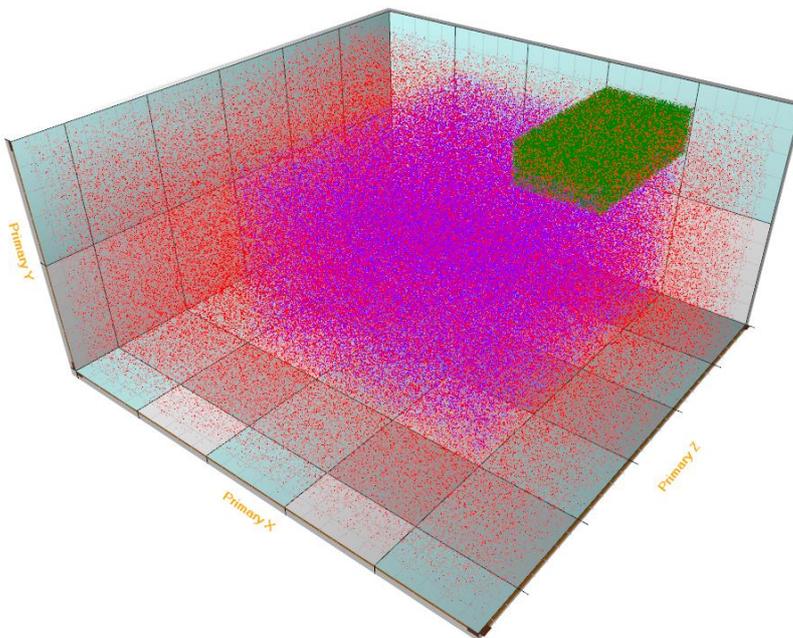


Figure 6-28. Millions of scatter points. LineVisible = False, PointsVisible = True, PointsOptimization = Pixels.

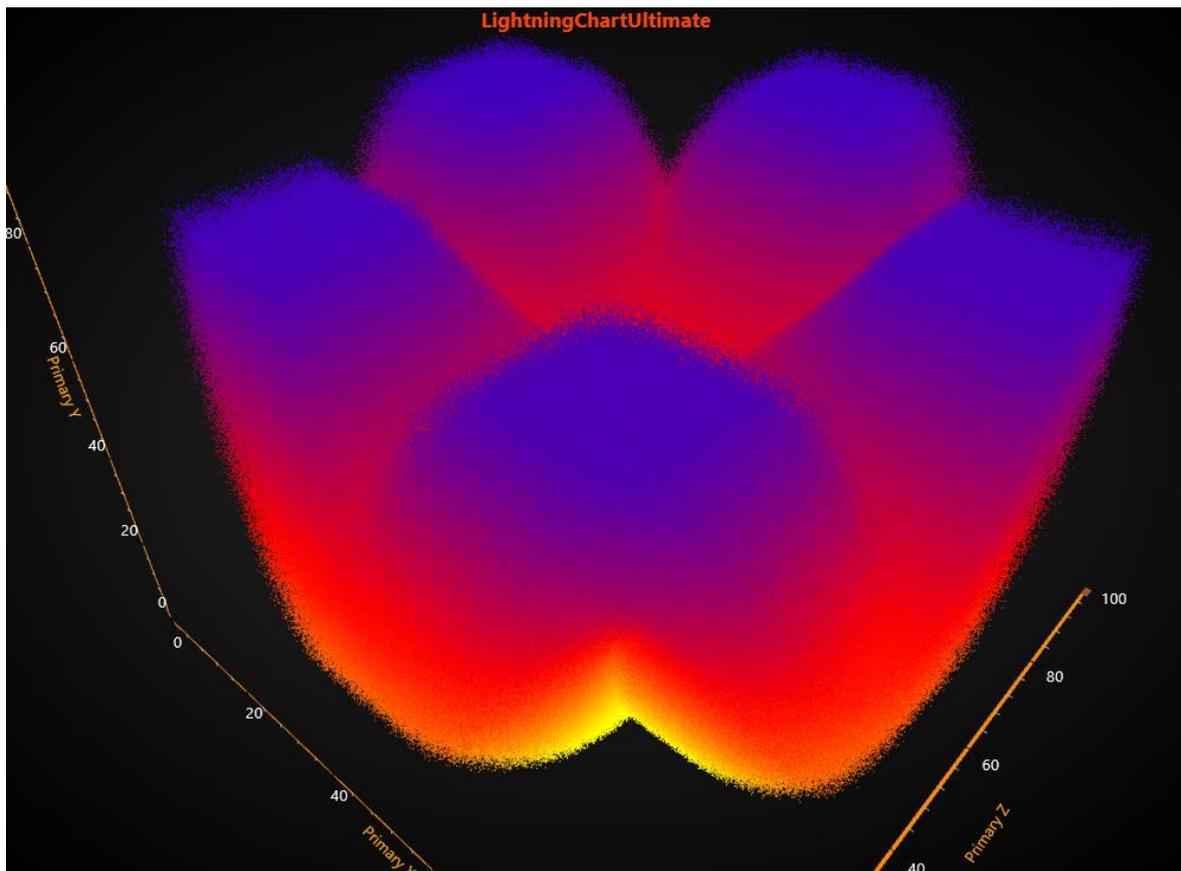


Figure 6-29. `IndividualPointsColoring = True`, using `PointsCompactColored`, `LineVisible = False`, `PointsVisible = True`. 120 million of scatter points.

Millions of data points can be most efficiently visualized with rectangles. When using ***PointsCompactColored*** or ***PointsCompact***, the point size can be controlled with ***PointStyle.Shape2D.Width*** and ***PointStyle.Shape2D.Height***.

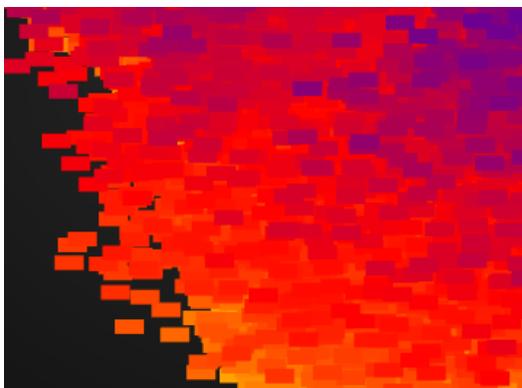


Figure 6-30. `PointStyle.Shape2D.Width = 20` and `PointStyle.Shape2D.Height = 10`.

6.10 SurfaceGridSeries3D

SurfaceGridSeries3D allows visualizing data as a 3D surface. In **SurfaceGridSeries3D**, nodes are equally spaced in X dimension, and in Z dimension as well.

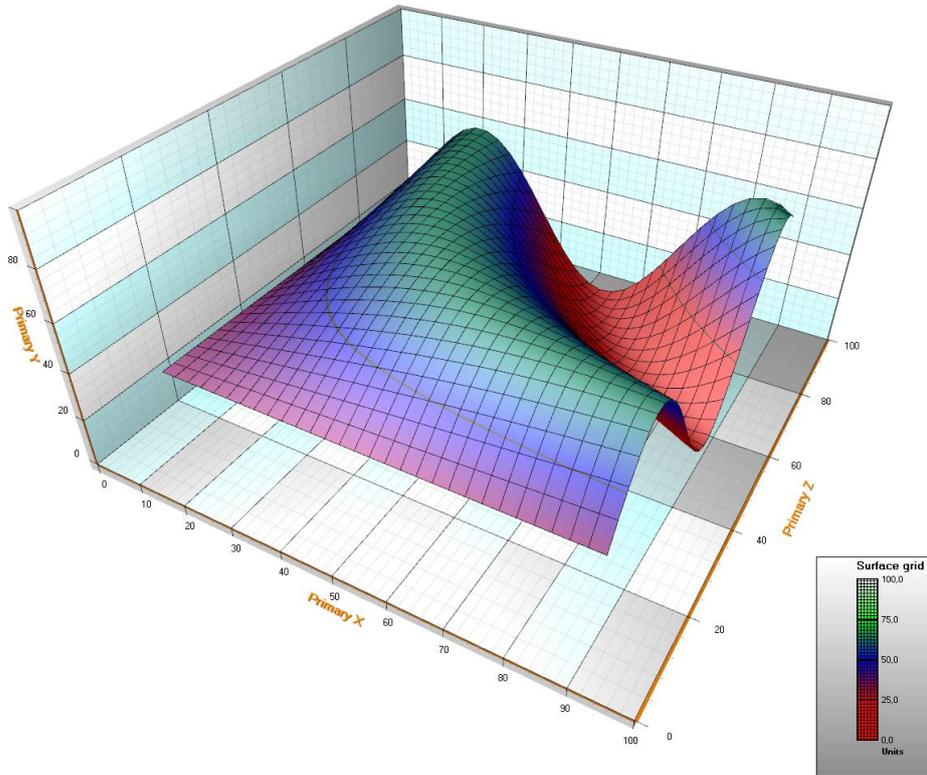


Figure 6-31. Surface grid series with default style. Height data is made with a sine formula. Legend box shows the height coloring intervals.

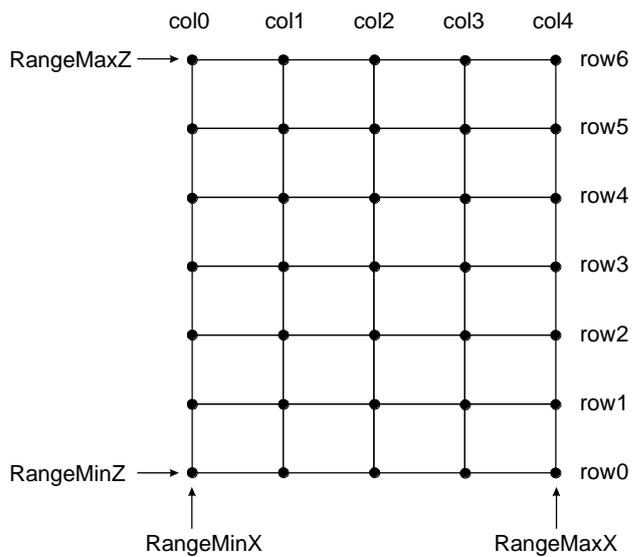


Figure 6-32. Surface grid nodes. SizeX = 5, SizeZ = 7.

Node distances are automatically calculated as

$$\text{node distance } X = \frac{\text{RangeMaxX} - \text{RangeMinX}}{\text{SizeX} - 1}$$

$$\text{node distance } Z = \frac{\text{RangeMaxZ} - \text{RangeMinZ}}{\text{SizeZ} - 1}$$

6.10.1 Setting surface grid data

- Set X range by using **RangeMinX** and **RangeMaxX** properties, to order the minimum and maximum value based on assigned X axis.
- Set Z range by using **RangeMinZ** and **RangeMaxZ** properties, to order the minimum and maximum value based on assigned Z axis.
- Set **SizeX** and **SizeZ** properties to give the grid a size as columns and rows.
- Set Y values for all nodes:

Method, with Data array index

```
for (int nodeIndexX = 0; nodeIndexX < columnCount; nodeIndexX ++)  
{  
    for (int nodeIndexZ = 0; nodeIndexZ < rowCount; nodeIndexZ ++)  
    {  
        Y //some height value.  
        gridSeries.Data[iNodeX, iNodeZ].Y = Y;  
    }  
}  
gridSeries.InvalidateData(); // Notify to refresh when the new values are ready
```

Alternative method, using SetDataValue

```
for (int nodeIndexX = 0; nodeIndexX < columnCount; nodeIndexX ++)  
{  
    for (int nodeIndexZ = 0; nodeIndexZ < rowCount; nodeIndexZ ++)  
    {  
        Y //some height value  
        gridSeries.SetDataValue(nodeIndexX, nodeIndexX,  
            0, //X value is irrelevant in grid  
            Y,  
            0, //Z value is irrelevant in grid  
            Color.Green); //Source point colors are not used in this  
            example, so use any color here  
    }  
}  
gridSeries.InvalidateData(); // Notify to refresh when the new values are ready
```

6.10.2 Creating surface from bitmap file

Surfaces can be created from bitmap images by using **SetHeightDataFromBitmap** method. The surface gets the size of the bitmap (if no anti-aliasing or resampling is used). For each bitmap image pixel, Red, Green and Blue values are summed. The greater the sum, the higher will be the height data value for that node. Black and dark colors get lower values while bright and white colors get higher values.

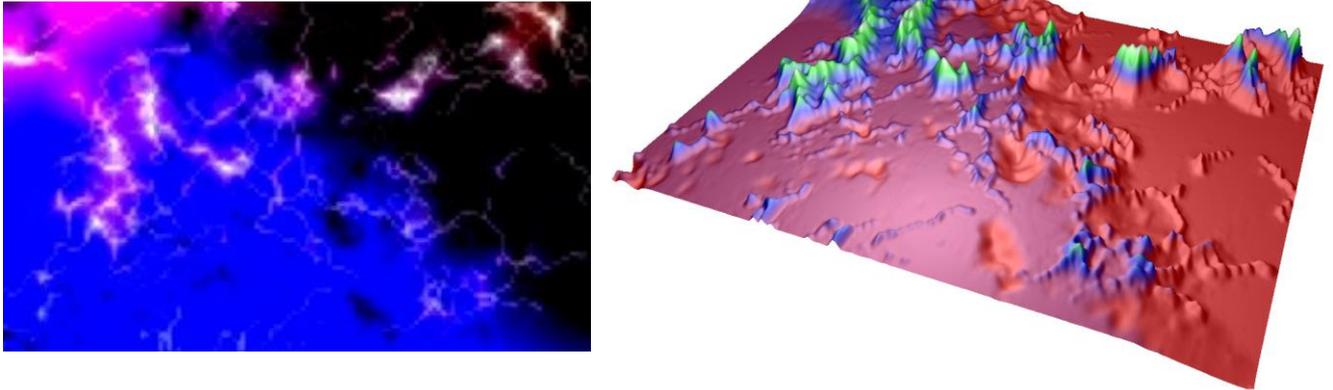


Figure 6-33. Source bitmap and calculated surface height data. Dark values stay low while bright values get higher in the surface.

6.10.3 Fill styles

Use **Fill** property to select the filling style of the surface. The following options are available:

- **None**: By using this, no filling is applied. This selection is useful with wireframe meshes.
- **FromSurfacePoints**: The colors of the Data property nodes are used.
- **Toned**: ToneColor applies
- **PalettedByY**: Coloring by Y values by palette, see chapter 6.10.4.
- **PalettedByValue**: Coloring by **SurfacePoint's Value** fields by palette, see chapter 6.10.4.
- **Bitmap**: Bitmap image is stretched to cover the whole surface. Set the bitmap image in **BitmapFill** property. **BitmapFill** property has sub-properties to mirror the image vertically and horizontally.

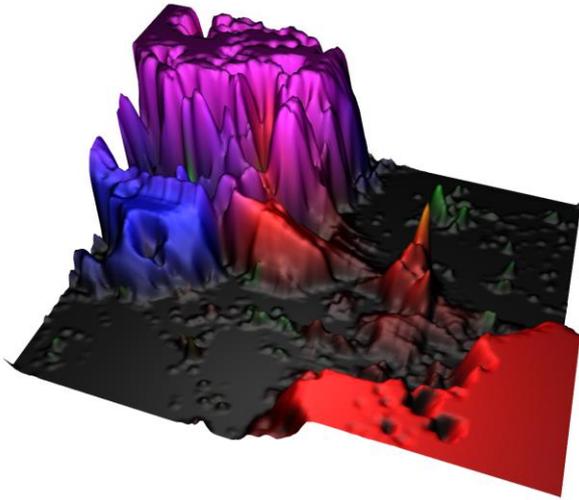


Figure 6-19. FromSurfacePoints fill. Color per data point.

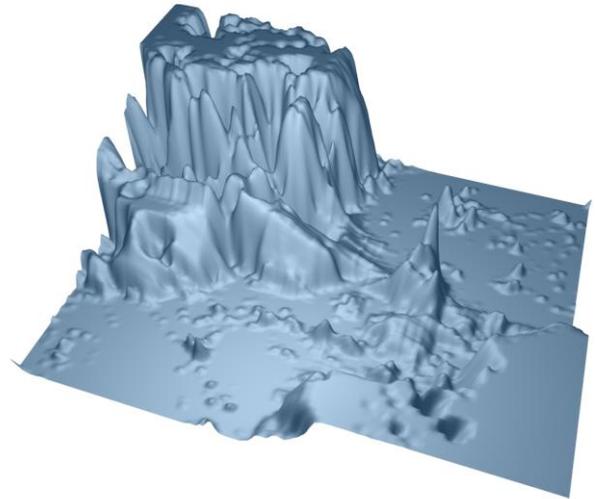


Figure 6-35. Toned fill.

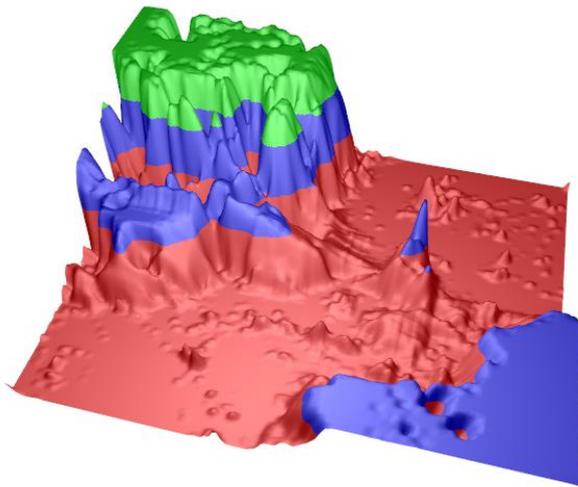


Figure 6-20. PalettedByY

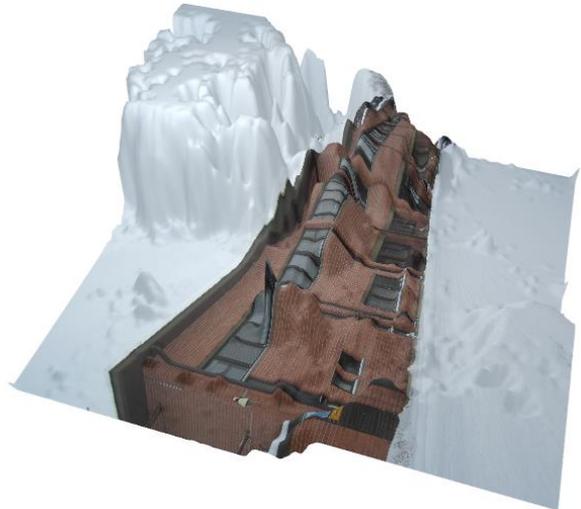


Figure 6-37. Bitmap fill.

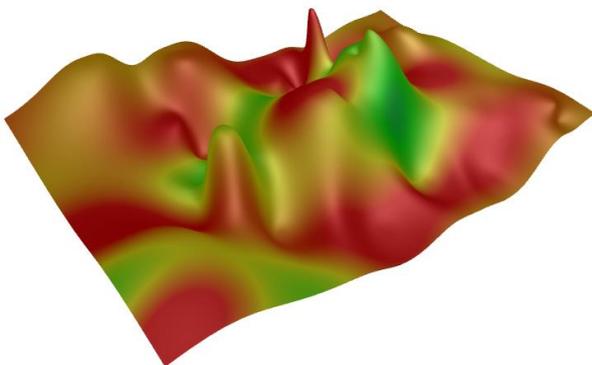


Figure 6-38. PalettedByValue.

6.10.4 Contour palette

ContourPalette property allows defining color steps for height coloring. **ContourPalette** can be used for:

- **Fill** (see chapter 6.10.3)
- **Wireframe mesh** (see chapter 6.10.5)
- **Contour lines** (see chapter 6.10.6)

An unlimited count of steps can be defined for contour palette. Each step has a height value and a corresponding color.

The palette includes **MinValue**, **Type** and **Steps** properties. For **Type**, there are two choices: **Uniform** and **Gradient**. The contour palette of figure 6-39 shows:

- **MinValue:** 0
- **Type:** Gradient
- **Steps:**
 - Steps[0]: MaxValue: 25, Color: Red
 - Steps[1]: MaxValue: 50, Color: Blue
 - Steps[2]: MaxValue: 75, Color: Lime
 - Steps[3]: MaxValue: 100, Color: White

The height values below first step value are colored with first step's color.

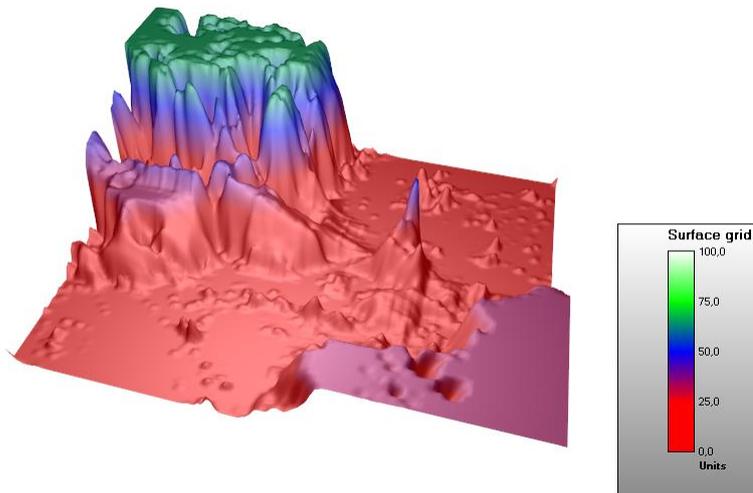


Figure 6-39. Surface grid series contour palette Type is set to Gradient.

6.10.5 Wireframe mesh

Use **WireframeType** to select the wireframe style. The options are:

- **None**: no wireframe
- **Wireframe**: a solid color wireframe. Use **WireframeLineStyle.Color** to set the color.
- **WireframePalettedByY**: wireframe coloring follows SurfacePoint's **Y** field **ContourPalette** (see chapter 6.10.4)
- **WireframePalettedByValue**: the wireframe coloring follows SurfacePoint's **Value** field, **ContourPalette** (see chapter 6.10.4)
- **WireframeSourcePointColored**: the wireframe coloring follows the color of the surface nodes
- **Dots**: solid color dots are drawn in the node positions
- **DotsPalettedByY**: dots are drawn in the node positions, and colored by **ContourPalette** according to **Y** field of SurfacePoints
- **DotsPalettedByValue**: dots are drawn in the node positions, and colored by **ContourPalette** according to **Value** field of SurfacePoints
- **DotsSourcePointColored**: dots are drawn in the node positions, coloring follows the color of the surface nodes

Wireframe line style (**color, width, pattern**) can be edited via **WireframeLineStyle**.

Note! Palette colored wireframe lines and dots are available only when **WireframeLineStyle.Width = 1** and **WireframeLineStyle.Pattern = Solid**.

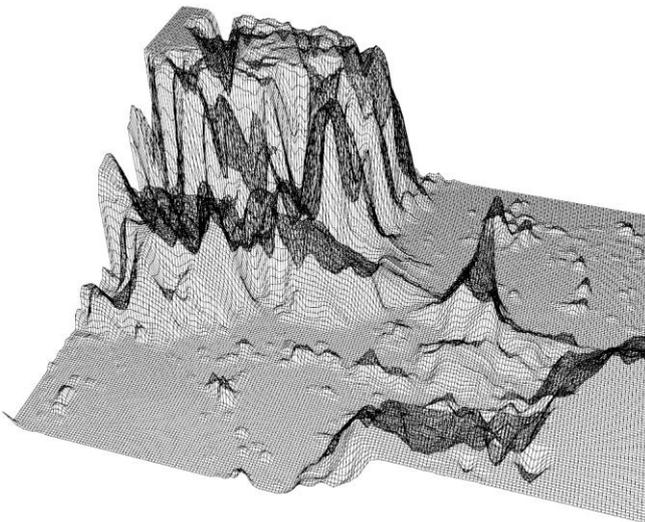


Figure 6-40. WireframeType = Wireframe.

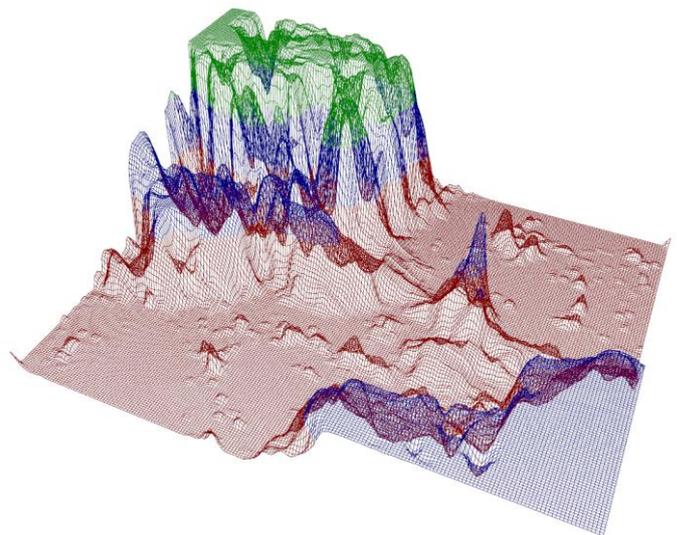


Figure 6-41. WireframeType = WireframePalettedByY.

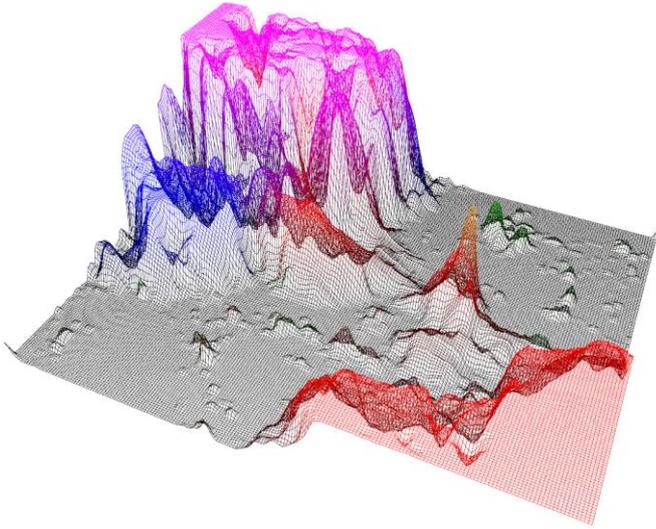


Figure 6-21. WireframeType = SourcePointColored.

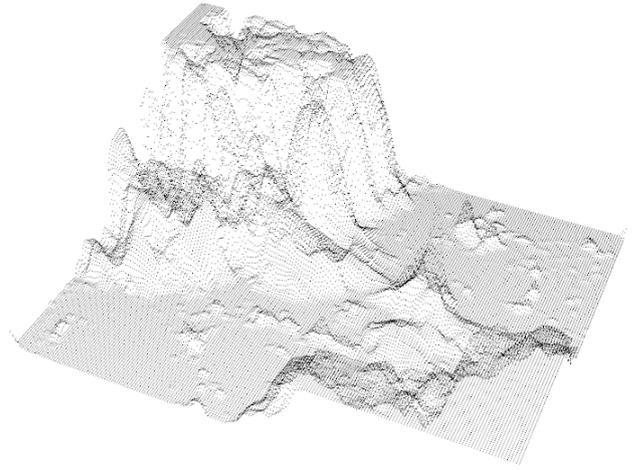


Figure 6-43. WireframeType = Dots.

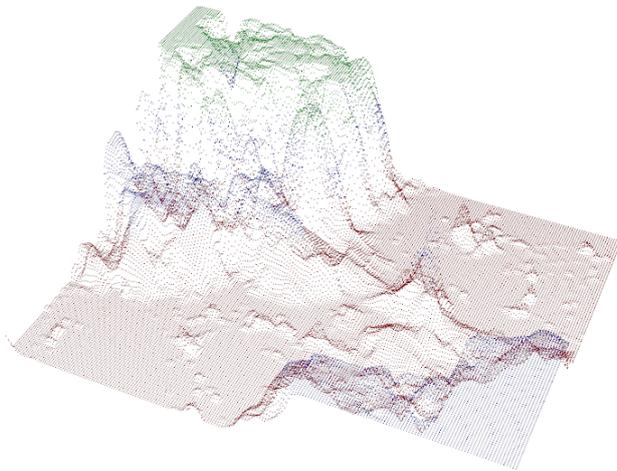


Figure 6-44. WireframeType = DotsPalettedByY.

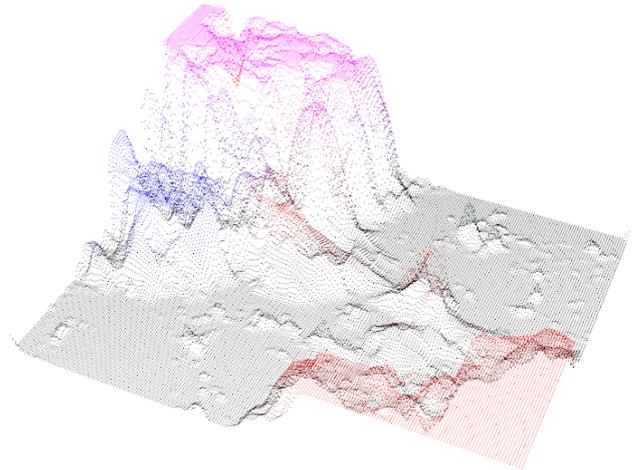


Figure 6-45. WireframeType = DotsSourcePointColored.

6.10.5.1 Some notes when using wireframe simultaneously with fill

When fill and wireframe are drawn in the same position in 3D model, *Z-fighting* may appear. It can be seen as broken wireframe lines. That is because it's impossible for the GPU to determine which object is closer to camera.

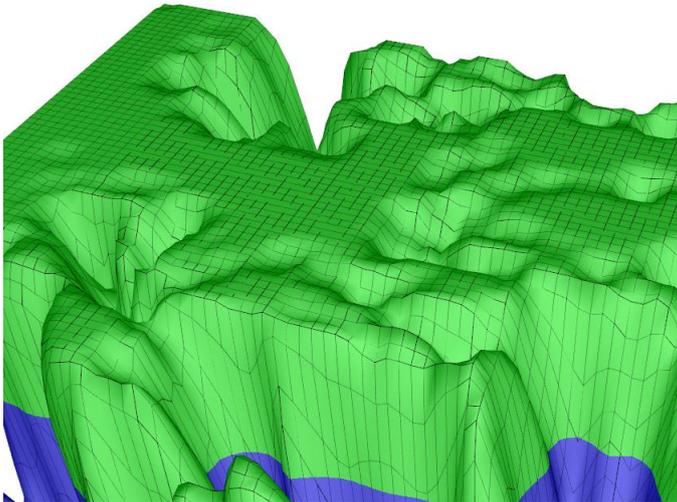


Figure 6-46. Surface grid wireframe with filling. Z-fighting appears as broken wireframe lines.

To prevent Z-fighting from occurring, use **WireframeOffset** or **DrawWireframeThrough** property. By using **WireframeOffset**, the wireframe is moved slightly in 3D model space. **DrawWireframeThrough** draws the wireframe through the filling, whether or not the part of the surface is visible to the camera.

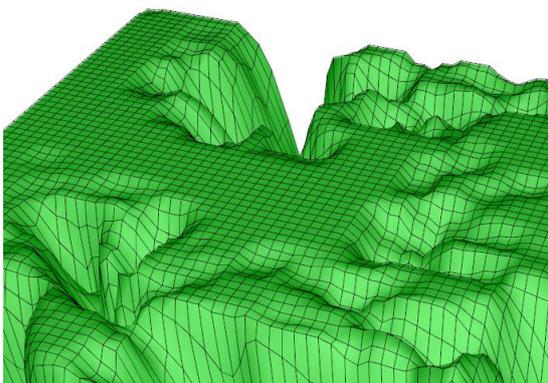


Figure 6-47. WireframeOffset = (X=0; Y=0.1; Z=0).

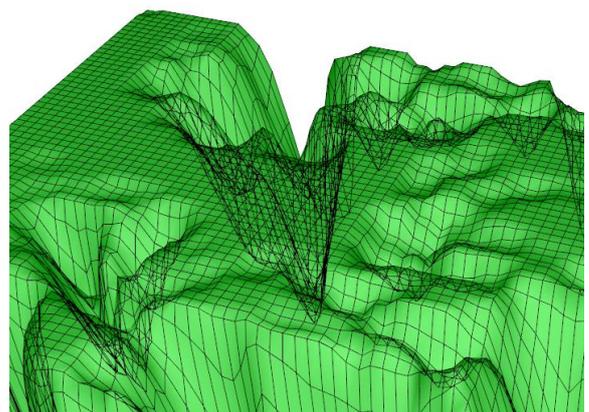


Figure 6-48. DrawWireframeThrough is enabled.

6.10.6 Contour lines

Contour lines allow quick interpretation of height data without filling the surface with paletted fill. Contour lines can be used combined with fill and wireframe. By setting **ContourLineStyle** property, contour lines can be drawn with different styles:

- **None**: no contour lines are shown
- **FastColorZones**: The lines are drawn as thin vertical zones. Allows very powerful rendering, which suits well for continuously updated or animated surface. Steep height changes are shown as thin line, as gently sloping height differences are shown with thick line. All lines use the same color defined with **ContourLineStyle.Color** property. The zone height can be set by **FastContourZoneRange** property.
- **FastPalettedZones**: Like **FastColorZones**, but line coloring follows **ContourPalette** options (see chapter 6.10.4).
- **ColorLineByY** and **ColorLineByValue**: Contour lines are made with actual lines. Rendering takes longer than **FastColorZones**. The line width can be adjusted with **ContourLineStyle.Width** property. Contour lines can also be shifted with **WireframeOffset** property, to remove possible Z-fighting with filling.
- **PalettedLineByY** and **PalettedLineByValue**: Like **ColorLineByY** and **ColorLineByValue**, but line coloring follows **ContourPalette** options (see chapter 6.10.4).

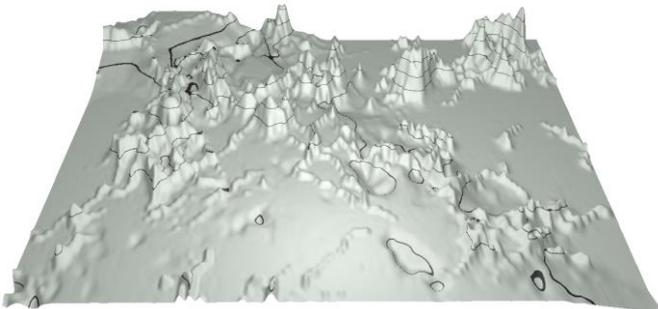


Figure 6-49. ContourLineStyle = FastColorZones.

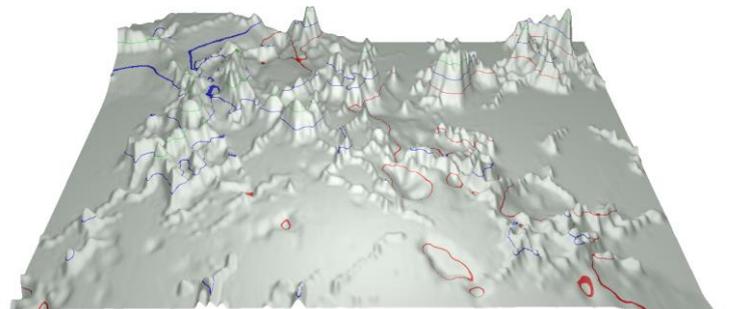


Figure 6-50. ContourLineStyle = FastPalettedZones.

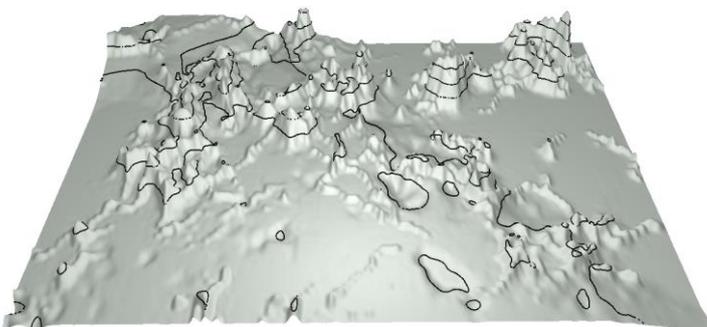


Figure 6-51. ContourLineStyle = ColorLine.

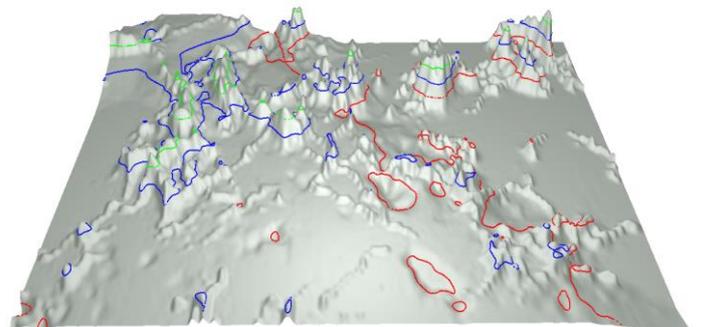


Figure 6-52. ContourLineStyle = PalettedLine.

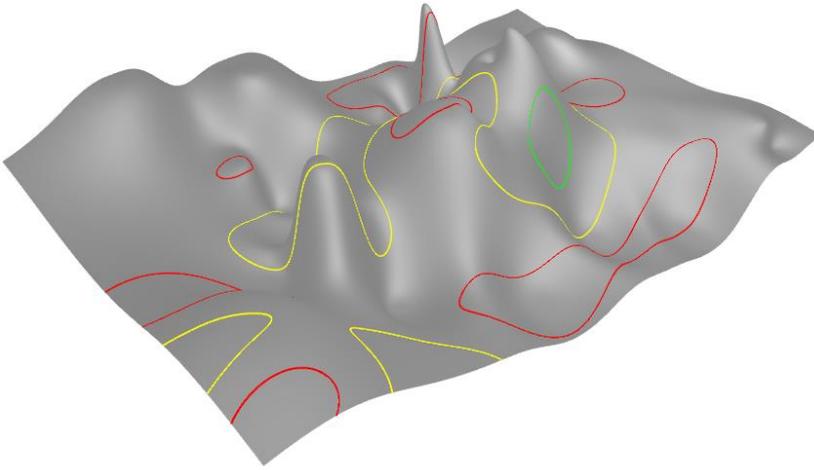


Figure 6-53. `ContourLineType = PalettedLineByValue`.

6.10.7 Fadeaway

SurfaceGridSeries3D, *SurfaceMeshSeries3D* and *WaterfallSeries3D* have a **FadeAway** property, which allows fading away the series towards the back of the chart. **Fadeaway** is measured in percents, valid range being from 0 (no fadeaway, the default value) to 100 (full fadeaway). The higher the value, the more transparent the data with high Z value will become.

6.10.8 Scrolling surface data

SurfaceGridSeries3D and *SurfaceMeshSeries3D* have **InsertRowBackAndScroll** and **InsertColumnBackAndScroll** methods for performance optimized periodical data adding. They insert a new data row or column into the surface series' data table while dropping the oldest values i.e. the first data row off. Consider the following 3D spectrum display (Figure 6-53). New FFT values are added as a new row (close to camera), and the old data and the time axis (Z axis) must be scrolled. The oldest surface values must be dropped off.

InsertRowBackAndScroll and **InsertColumnBackAndScroll** take the new data as a double array, but also require new minimum and maximum values for both surface series and the scrolled axis (Z dimension/axis for **InsertRowBackAndScroll** and X dimension/axis for **InsertColumnBackAndScroll**). This constant adjusting of series and axis ranges enables the scrolling effect.

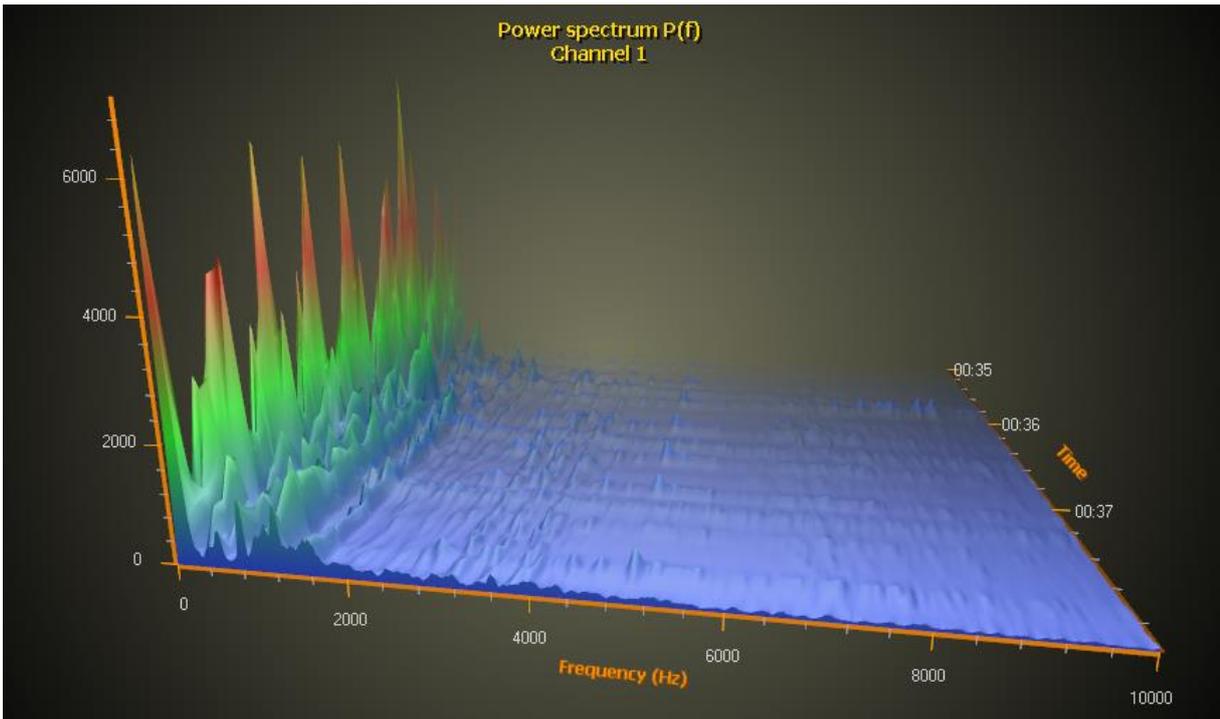


Figure 6-54. Presenting 3D spectrum with surface grid. InsertRowBackAndScroll method is used for performance optimized data adding. Fadeaway property is 100 to make the surface smoothly fade away towards the back of the chart. A perspective camera is used.

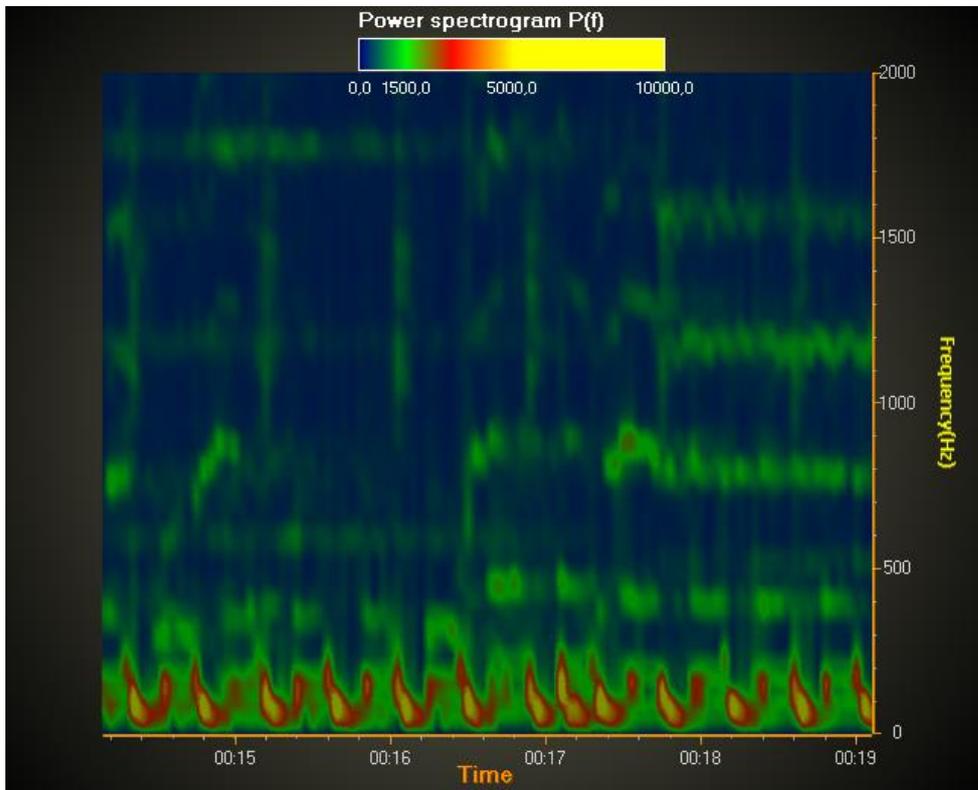


Figure 6-55. A spectrogram with surface grid. InsertColumnBackAndScroll method is used. An orthographic camera above the model gives straight and perpendicular projection. SuppressLighting is enabled to remove unwanted light reflections. Fadeaway = 0 for making the grid series fully visible.

6.11 SurfaceMeshSeries3D

SurfaceMeshSeries3D is almost similar to **SurfaceGridSeries3D** as they both mostly have the same properties. The biggest difference is that surface nodes can be positioned freely in 3D space. In other words, the surface does not have to be rectangular. **SurfaceMeshSeries3D** allows warping the surface virtually to any shape, for example to a sphere or a human head.

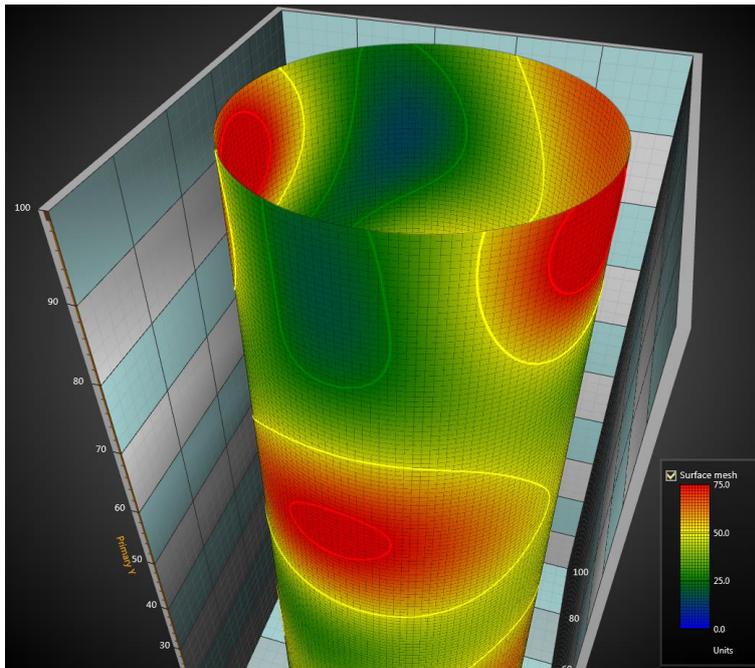


Figure 6-56. SurfaceMeshSeries3D, geometry made as a pipe.

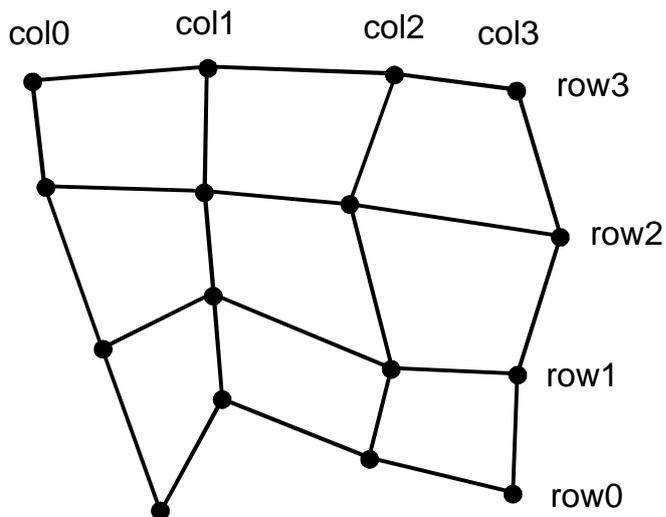


Figure 6-57. Surface mesh nodes. SizeX = 4, SizeZ =4.

6.11.1 Setting surface mesh data

- Set **SizeX** and **SizeZ** properties to give the grid a size as columns and rows.
- Set X, Y and Z values for all nodes:

Method, with Data array index

```
for (int nodeIndexX = 0; nodeIndexX < columnCount; nodeIndexX ++)  
{  
    for (int nodeIndexZ = 0; nodeIndexZ < rowCount; nodeIndexZ ++)  
    {  
        meshSeries.Data[nodeIndexX, nodeIndexZ].Y = xValue;  
        meshSeries.Data[nodeIndexX, nodeIndexZ].Y = yValue;  
        meshSeries.Data[nodeIndexX, nodeIndexZ].Z = zValue;  
        meshSeries.Data[nodeIndexX, nodeIndexZ].Value = dataValue;  
    }  
}  
meshSeries.InvalidateData(); //Notify new values are ready to refresh
```

Alternative method, usage of SetDataValue

```
for (int nodeIndexX = 0; nodeIndexX < columnCount; nodeIndexX ++)  
{  
    for (int nodeIndexZ = 0; nodeIndexZ < rowCount; nodeIndexZ ++)  
    {  
        meshSeries.SetDataValue(nodeIndexX, nodeIndexZ,  
            xValue,  
            yValue,  
            zValue,  
            dataValue,  
            Color.Green); //Source point colors are not used in this  
                           example, so use any color here  
    }  
}  
meshSeries.InvalidateData(); //Notify new values are ready to refresh
```

6.12 WaterfallSeries3D

With **WaterfallSeries3D**, the data is visualized in area strips. Areas can be filled, wire-framed and contour-lined like **SurfaceGridSeries3D**, see chapter 6.10. In Y-dimension, area starts from **BaseLevel** property value. The node data can be set like in **SurfaceMeshSeries3D**, see chapter 6.11.1.

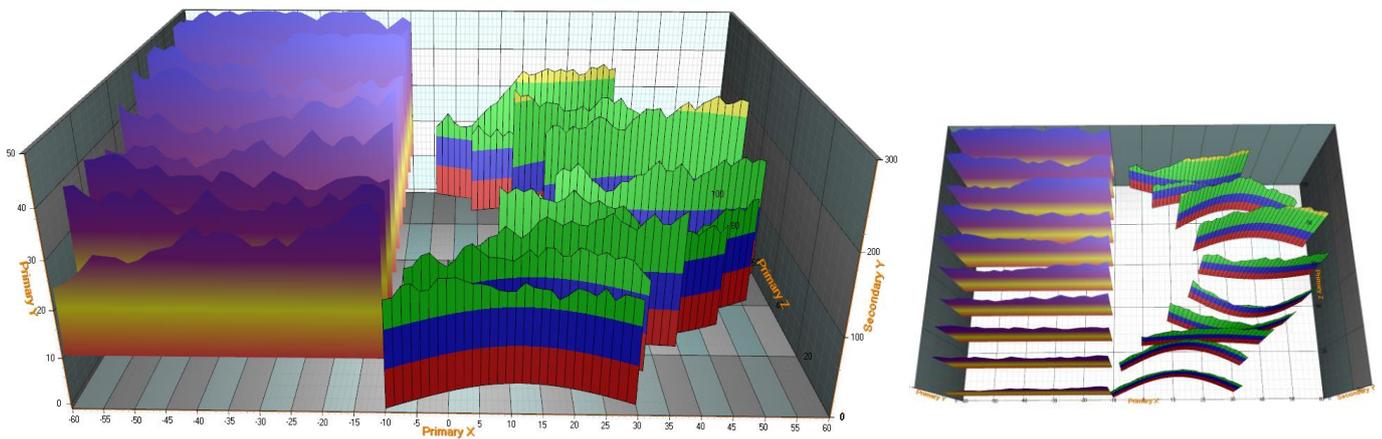


Figure 6-58. Two waterfall series. On the left violet series, X and Z are in rectangular form. BaseLevel = 10. On the right red-green-blue series, X and Z values are bent, and each row is placed in different horizontal location.

WaterfallSeries3D is especially handy for presenting traditional 3D spectrum.

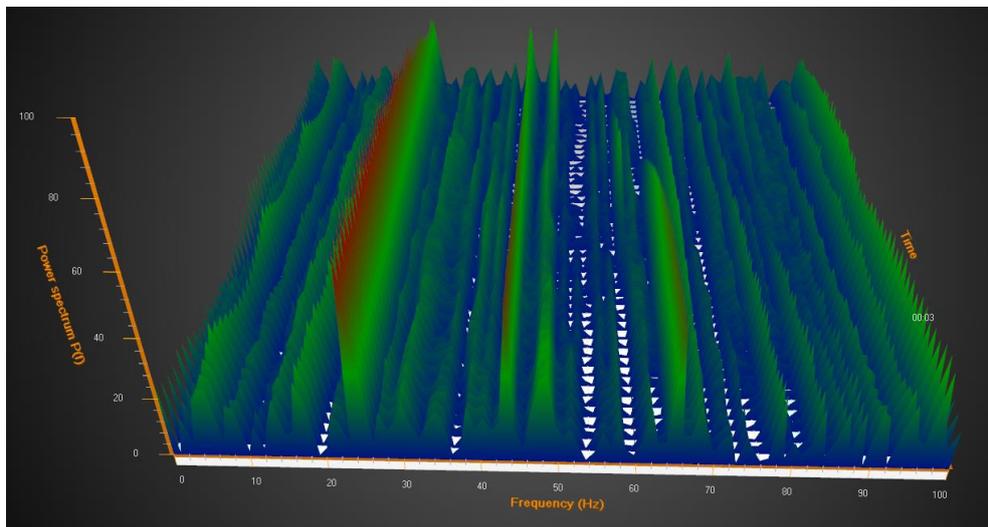


Figure 6-59. Waterfall series used for traditional spectrum presentation.

6.13 BarSeries3D

BarSeries3D allows bar data visualization in 3D.

6.13.1 Bars grouping

Bar series can be grouped with many options available in **BarViewOptions** property of View3D. **BarViewOptions.ViewGrouping** controls how the bars are grouped in the 3D view.

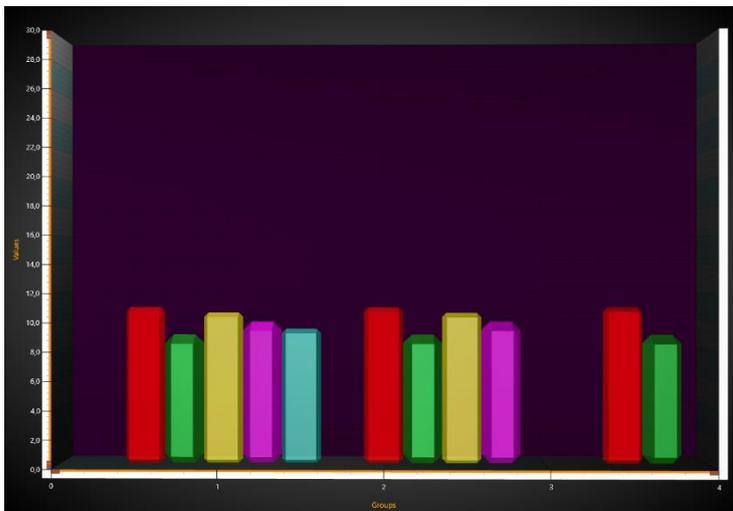


Figure 6-60. **BarViewOptions.ViewGrouping = GroupedIndexedFitWidth**. Bars are grouped according to their index. Bar widths and group gaps are arranged to fit the width nicely.

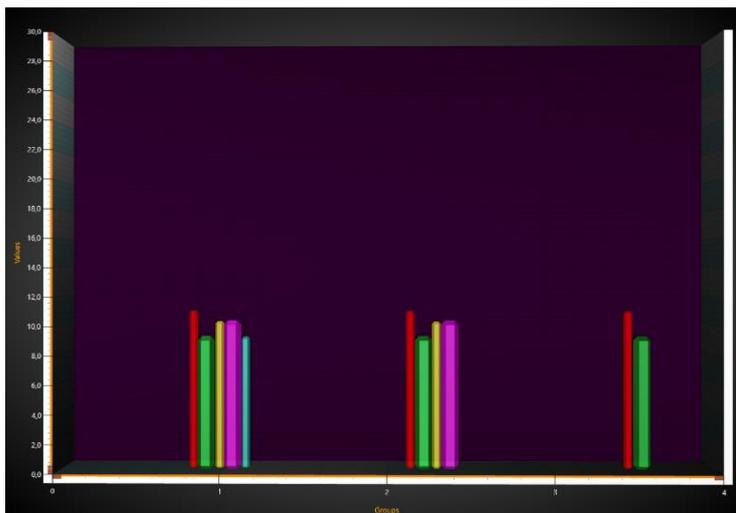


Figure 6-61. **BarViewOptions.ViewGrouping = GroupedIndexed**. Original bar widths apply, and groups are arranged to fit the chart width.

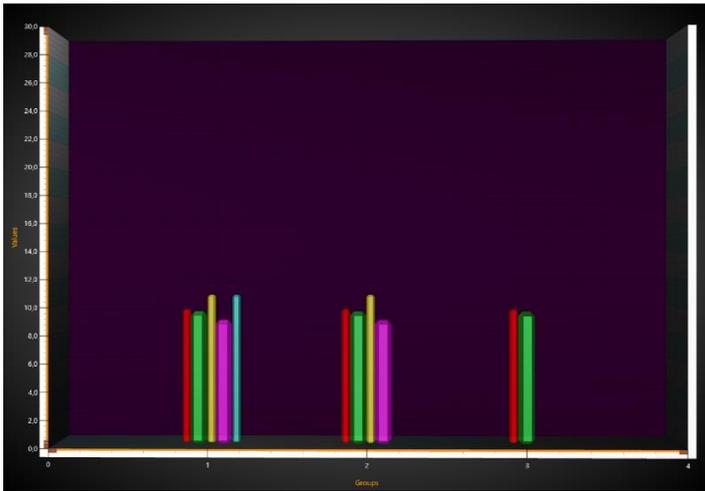


Figure 6-62. `BarViewOptions.ViewGrouping = GroupedByXValue`. Bar X values apply.

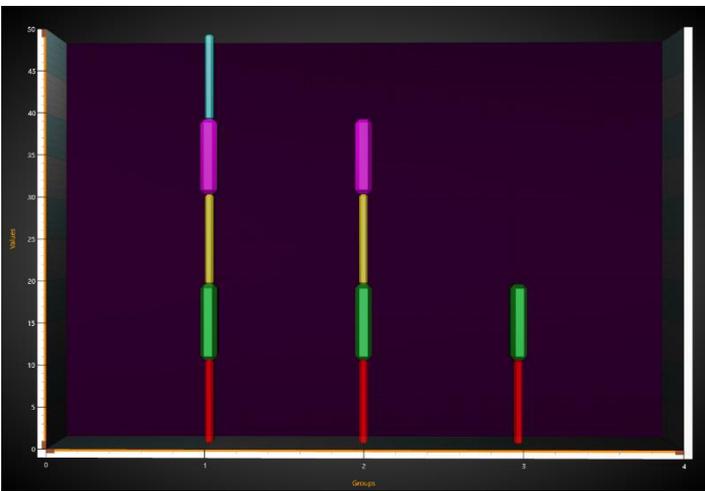


Figure 6-63. `BarViewOptions.ViewGrouping = StackedIndexed`. All bars having same index are stacked.

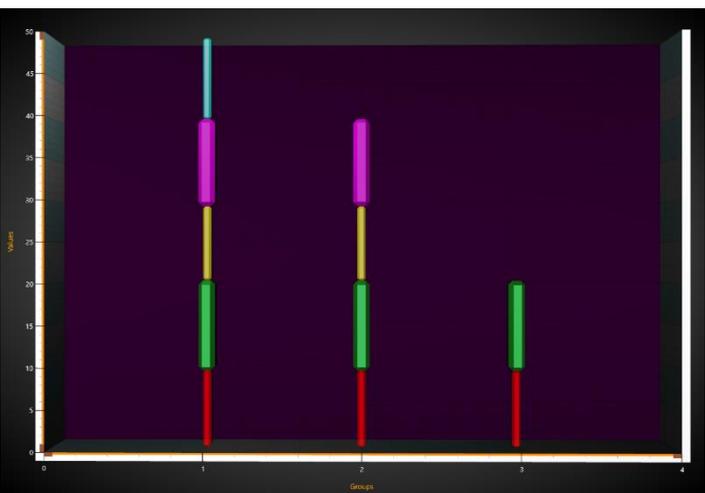


Figure 6-64. `BarViewOptions.ViewGrouping = StackedByXValue`. All bars having same X value are stacked. This example looks same than with `StackedIndexed`, as the X values and indices are same.



Figure 6-65. `BarViewOptions.ViewGrouping = StackedStretchedToSum`. All bars having same X value are stacked and stretched to `StackSum`, in this case 25.

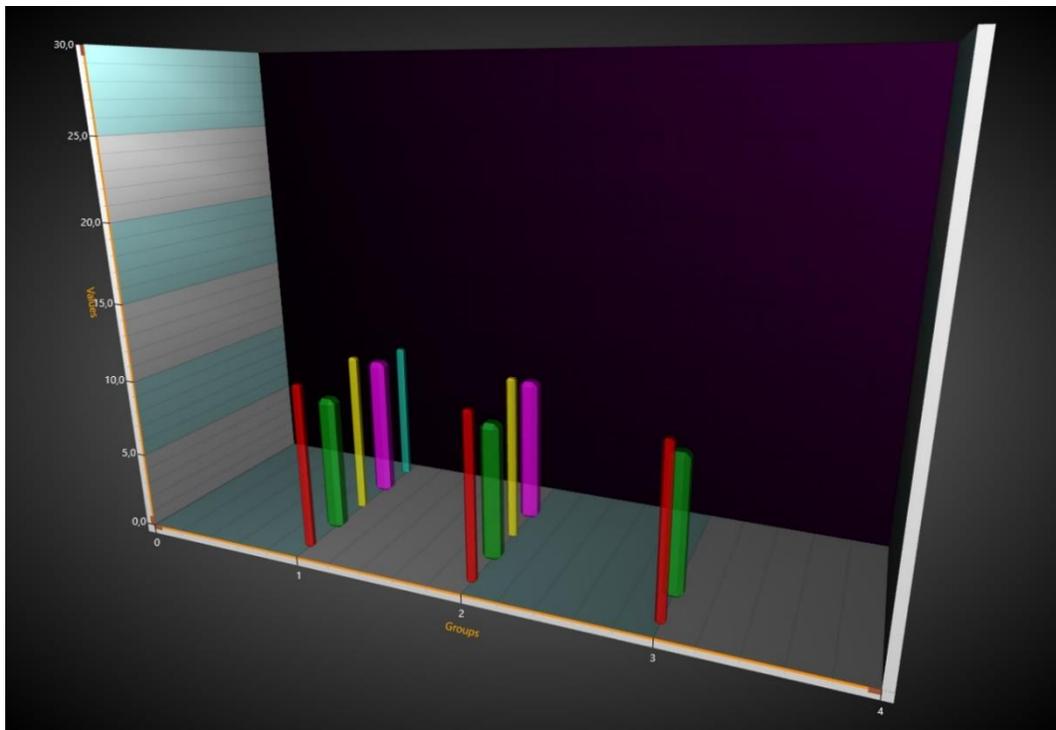


Figure 6-66. `BarViewOptions.ViewGrouping = Manhattan`. The first series values are shown nearest to the camera and the last series farthest. Bar X values control the bar position in X dimension.

6.13.2 Bar styles

BarSeries3D has **Shape** property for controlling the bar shape. In addition, with some shapes, **CornerPercentage** can be used to change corner rounding and **DetailLevel** to change the visual quality.

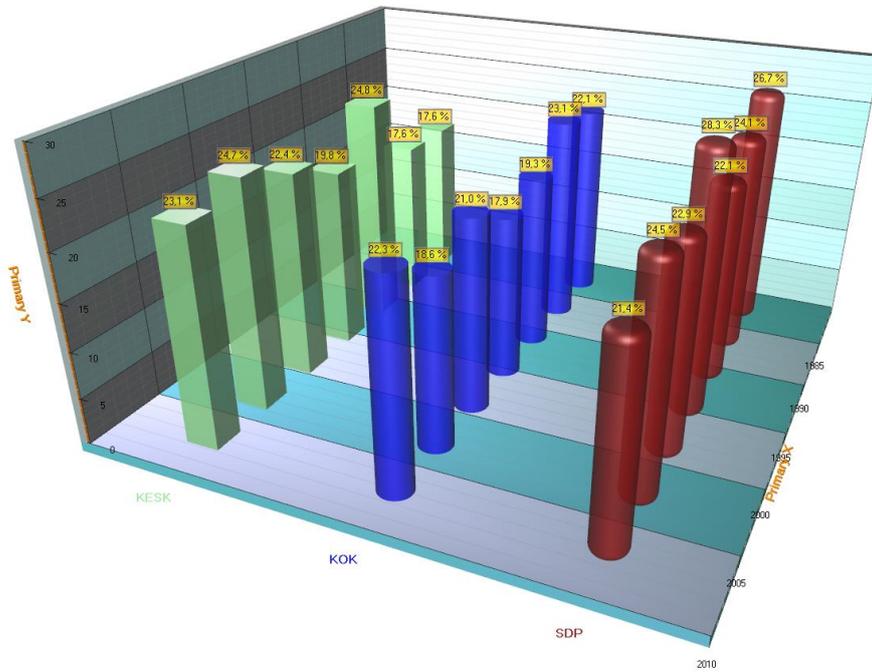


Figure 6-67. Bar shapes: Simple, Cylinder and RoundedCylinder.

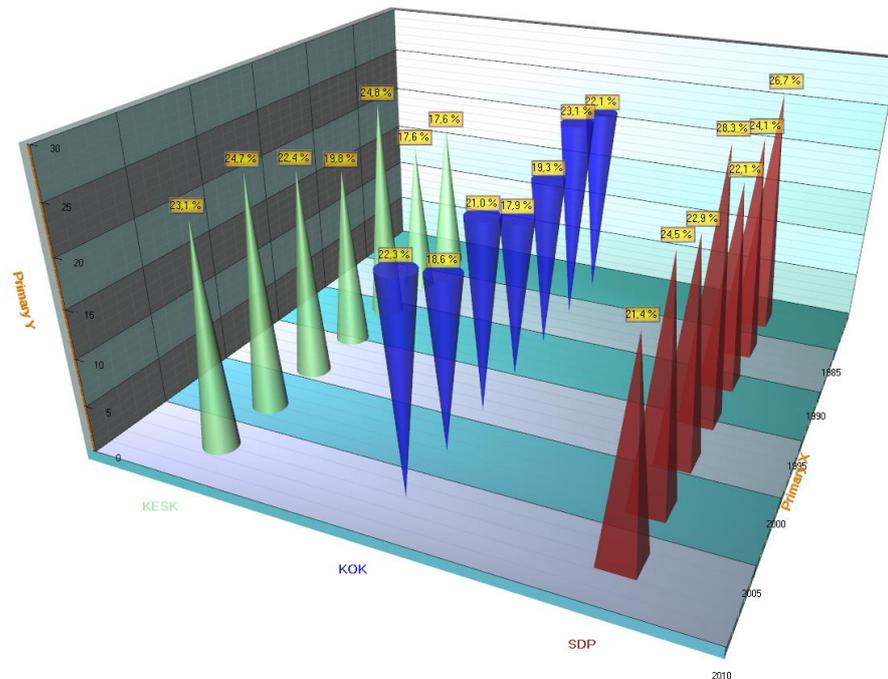


Figure 6-68. Bar shapes: Cone, ReversedCone and Pyramid.

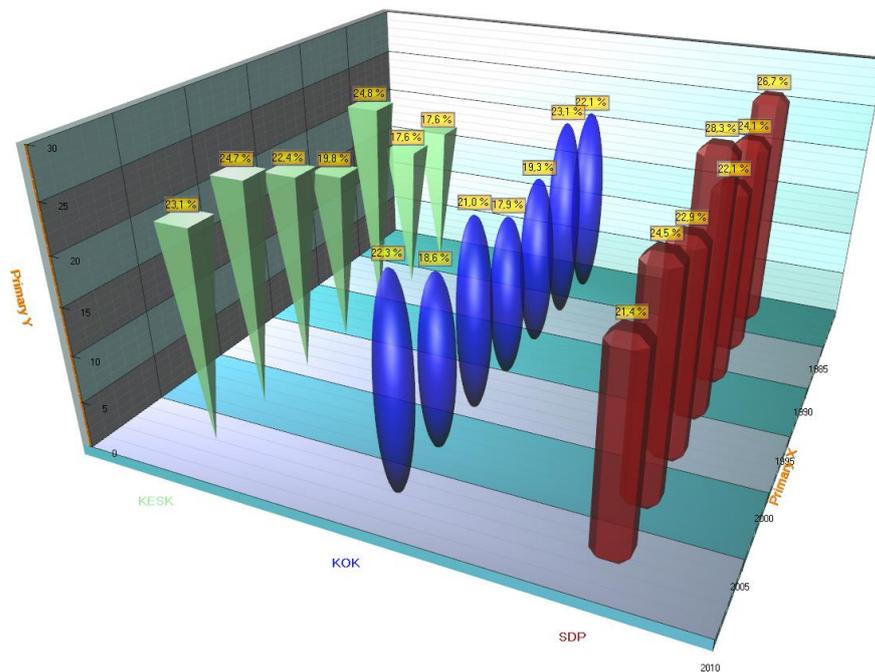


Figure 6-69. Bar shapes: ReversedPyramid, Ellipsoid and Beveled.

6.13.3 Setting bar series data

Bar series data can be added as **BarSeriesValue3D**-structures, which contains **x**, **y**, **z** and **text** fields.

```
// create new values array
BarSeriesValue3D[] values = new BarSeriesValue3D[3];
values[0] = new BarSeriesValue3D(20, 45, 5, "");
values[1] = new BarSeriesValue3D(30, 50, 5, "");
values[2] = new BarSeriesValue3D(40, 35, 5, "");

// add values to series
chart.View3D.BarSeries3D[0].AddValues(values, false);
```

6.13.4 Showing bars horizontally

Bars are drawn in Y axis direction. To show the bars vertically, rotate the camera 90 degrees.

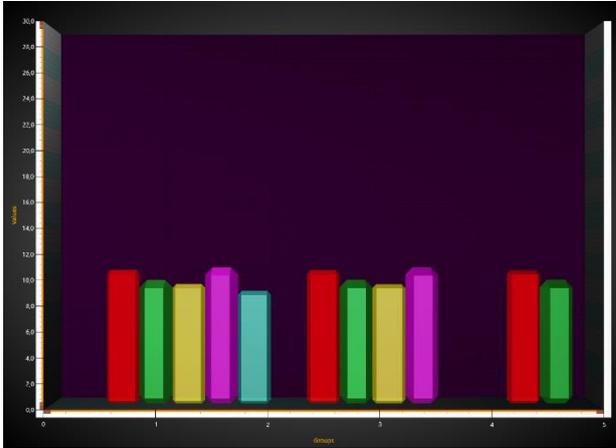


Figure 6-70. Vertical bars view.

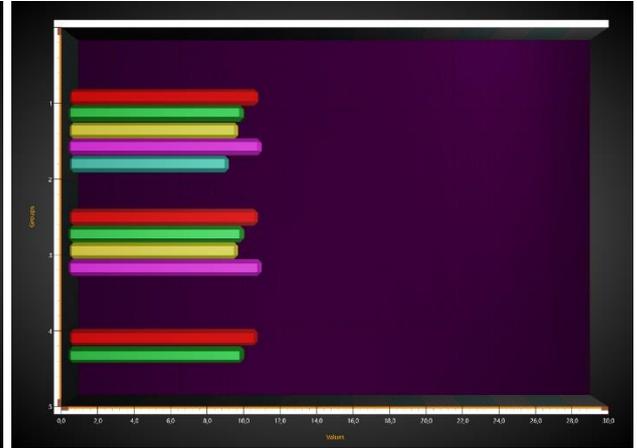


Figure 6-71. Horizontal bars view.

The code setting up the **vertical bars** view of the previous figure:

```
chart.BeginUpdate();
chart.View3D.Dimensions.Y = 100;
chart.View3D.Dimensions.X = 150;
chart.View3D.YAxisPrimary3D.Location = AxisYLocation3D.FrontLeft;
chart.View3D.Camera.RotationX = 0;
chart.View3D.Camera.RotationY = 0;
chart.View3D.Camera.RotationZ = 0;
chart.View3D.Camera.ViewDistance = 170;
chart.EndUpdate();
```

The code setting up the **horizontal bars** view of the previous figure:

```
chart.BeginUpdate();
chart.View3D.Dimensions.Y = 150;
chart.View3D.Dimensions.X = 100;
chart.View3D.YAxisPrimary3D.Location = AxisYLocation3D.FrontRight;
chart.View3D.Camera.RotationX = 0;
chart.View3D.Camera.RotationY = 0;
chart.View3D.Camera.RotationZ = 90;
chart.View3D.Camera.ViewDistance = 170;
chart.EndUpdate();
```

6.14 MeshModels

MeshModels list property allows inserting 3D models from external 3D model editors into LightningChart View3D. The models can be imported in OBJ format, which is a generic format in 3D modeling applications and game engines.

Note! LightningChart v.7 onwards does not support Direct3D X-format files (*.x) anymore, since DirectX 11 does not support it.

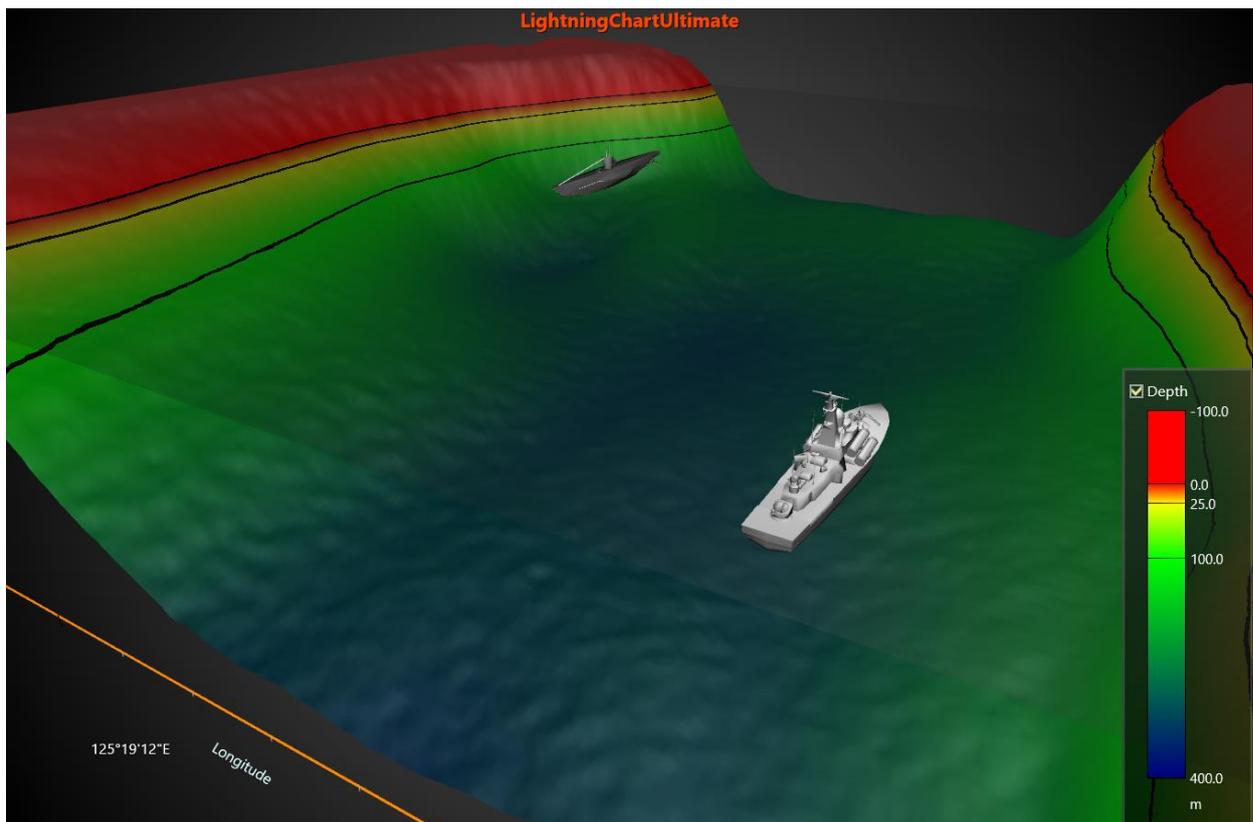


Figure 6-72. Battleship and submarine models loaded in View3D, over SurfaceGridSeries3D visualizing seabed depth data.

6.14.1 Loading a model

- To load a model from file, set the path and file name into **ModelFileName** property, or use **LoadFromFile** method. When loading the model from file, texture fills are loaded as well, if they exist in the same path, and MTL file and image files are accessible.
- To load model from stream, use **LoadFromStream** method. The stream reading method only reads geometry and materials, but not textures.
- To load model from a resource, use **LoadFromResource** method.

6.14.2 Positioning, scaling and rotating the model

A **MeshModel** object **Position** follows the X, Y and Z axes it has been assigned to. The model can be rotated by editing **Rotation** property. **Size** can be defined with **Size** property, which is a collection of factors for original model size and does not follow axis ranges or 3D world dimensions.

6.14.3 Enabling fill and wireframe

- To show fill, set **Fill** = True
- To show wireframe, set **WireFrame** = True, and set preferred line color in **WireFrameLineColor**.

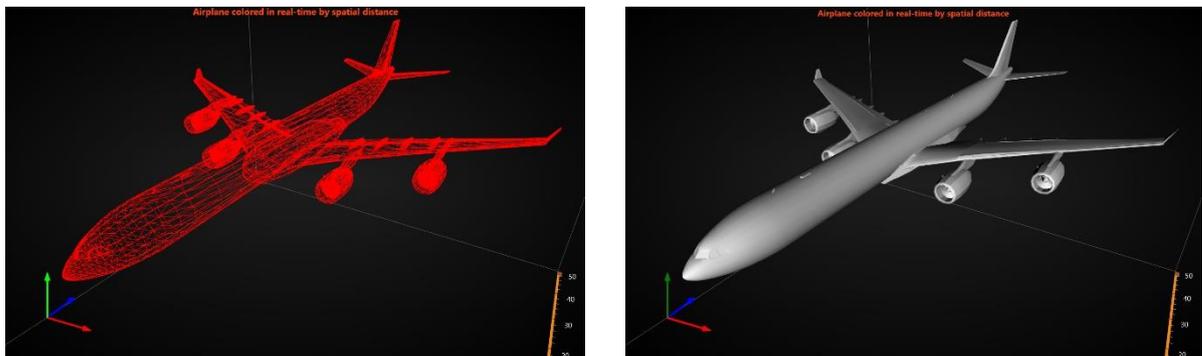


Figure 6-73. Airplane shown as wireframe (**WireFrameLineColor** = Red) and with default gray fill.

6.14.4 Custom-coloring fill

By default, the model renders with the colors of the OBJ model. To apply custom coloring for model's vertices, use **UpdateFillColor**(int[] colors) method. This method can also be called periodically, to apply real-time color updates.

GeometryConstructed event reports position of vertices in *axis values space*, as **X**, **Y** and **Z** arrays. They are especially needed when applying coloring e.g. by spatial distance of other chart objects, such as data points. Subscribe to **GeometryConstructed** event handler in the initialization phase, and then unsubscribe when not needed anymore.

UpdateFillColor requires ARGB colors array that is equal length of vertex positions (**X.Length**). One color for each vertex.

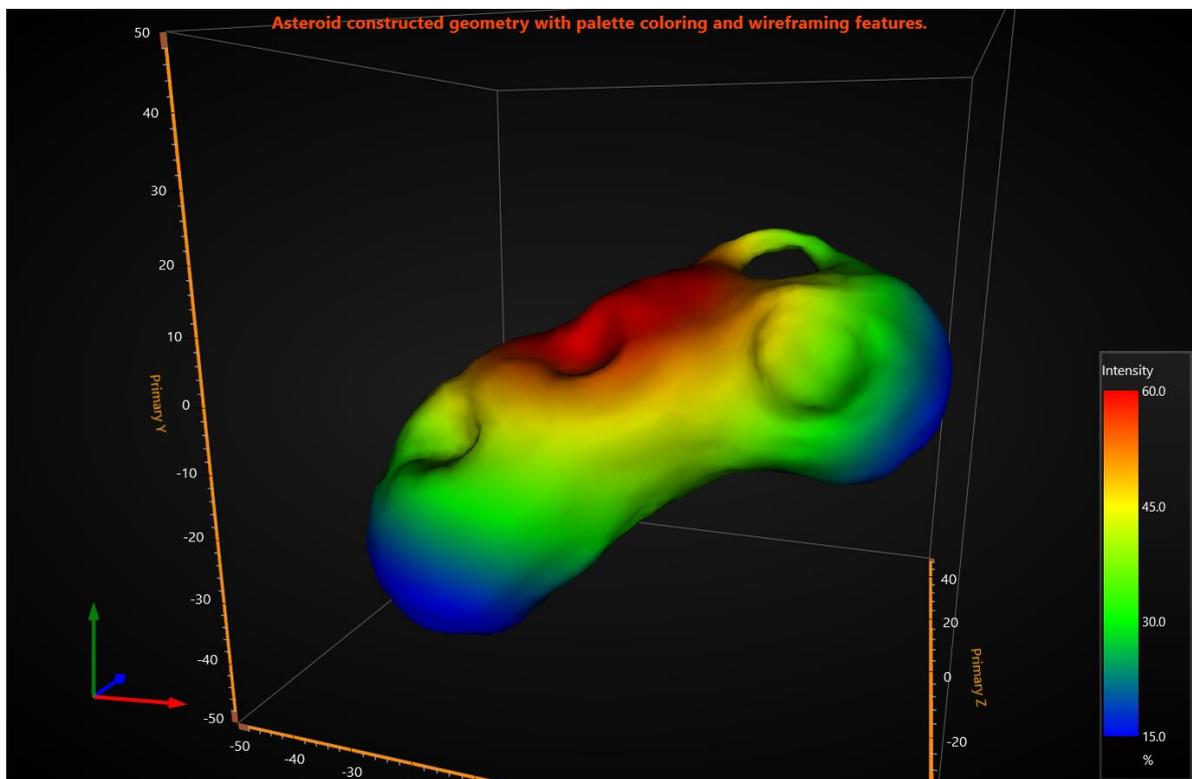


Figure 6-74. MeshModel colored by spatial distance utilizing UpdateFillColor method.

Note: **ChartTools.ConvertDataToColorsByFixedIntervalPalette** method can be utilized to convert data values into colors (ARGB int) by given palette steps.

6.14.5 Custom-coloring wireframe

Wireframe can also be colored with custom colors. Use **GeometryConstructed** event handler to get the required colors array length and **UpdateWireframeColors** method to apply the new colors.

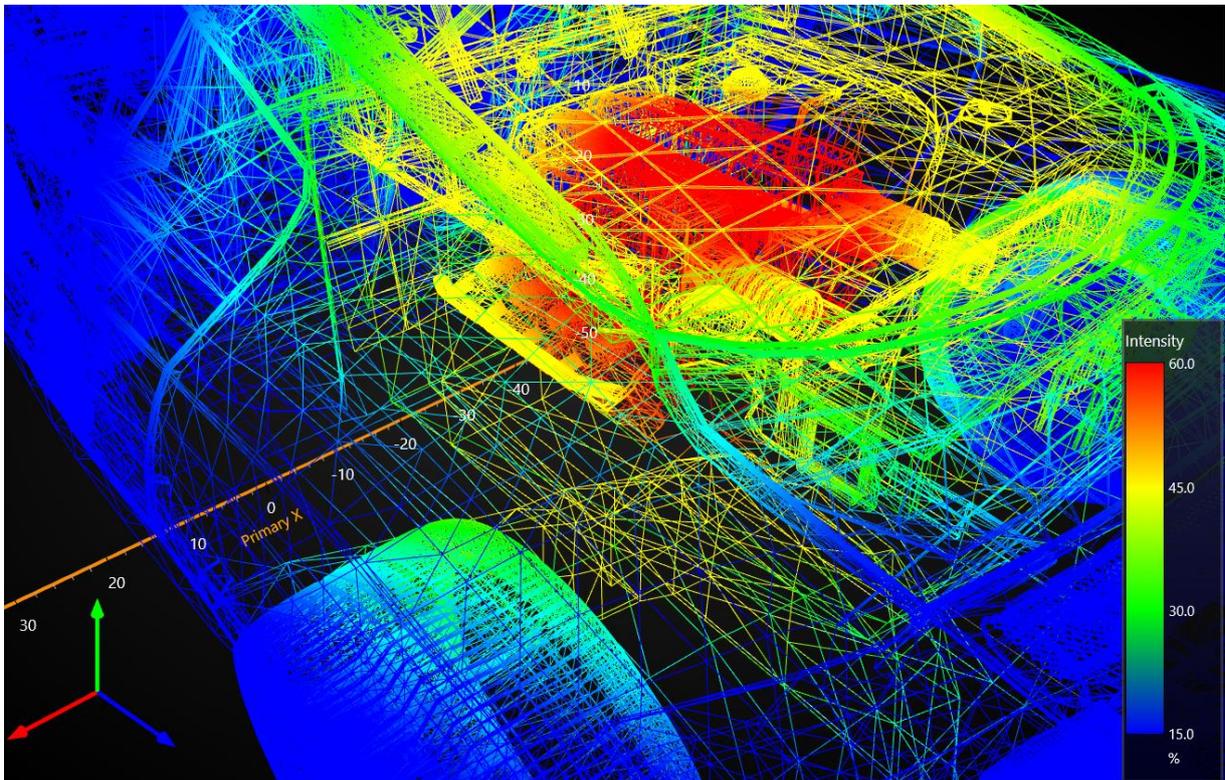


Figure 6-75. MeshModel wireframe colored by spatial distance utilizing UpdateWireframeColors method.

6.14.6 Reverse vertices winding order

Some models are made with reverse winding order and therefore culling makes them invisible. In case a model does not show up correctly, change **ClockwiseVertexWinding** setting.

6.14.7 Constructing MeshModel programmatically from vertices

Starting from v.8.2, **MeshModel** supports constructing the **MeshModel** geometry programmatically. It allows visualizing objects and shapes that have been produced via computation.

The following **Create** methods are available:

- `Create(positions, colors, indices)`
- `Create(positions, colors, normals, indices)`
- `Create(positions, textureCoordinates, bitmap, textureWrapMode, indices)`
- `Create(positions, normals, textureCoordinates, bitmap, textureWrapMode, indices)`

Index array (**indices**) parameters are optional. If provided, they will define which vertices, colors, light normals and texture coordinates to use from the arrays given. Using indices saves resources when same vertices are shared between multiple triangles.

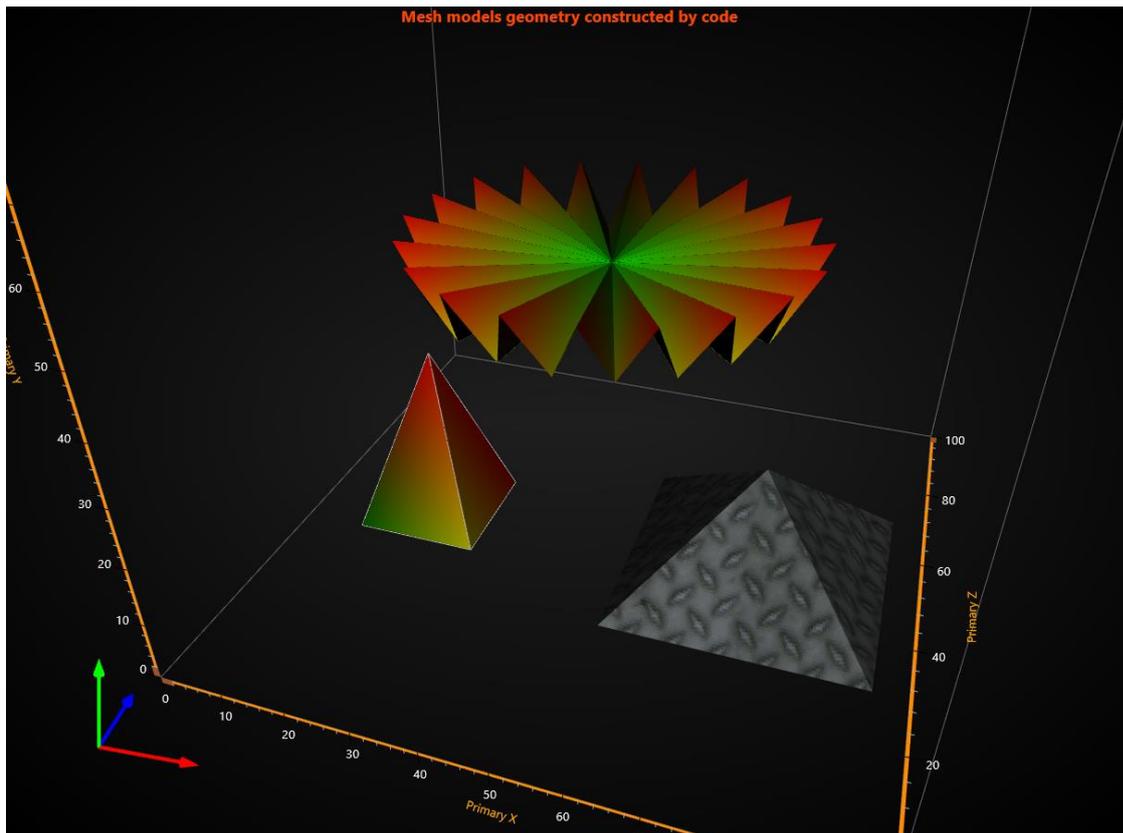


Figure 6-76. MeshModels constructed by code.

The rotation, scaling and positioning properties etc., as well as events, apply also to a **MeshModel** created programmatically from vertices, in a similar way than they work for loaded objects.

6.14.7.1 Updating the bitmap fill efficiently

When a **MeshModel** has been created by using **Create** method, supplying bitmap and texture coordinates as arguments, it is possible to update the bitmap very efficiently without reconstructing the geometry. Call **UpdateFillBitmap** method to update.

Note! **UpdateFillBitmap** method is not applicable for models loaded from OBJ files.

6.14.8 Tracing the model with mouse

MeshModel has triangle-based tracing for mouse position. Use **MouseTriangleTraced** event, which indicates the nearest triangle to the camera and the mouse location.

The event arguments have the following info:

- **IntersectionPointAxisValues**: intersection point of triangle face in axes values
- **ModelSpaceTriangleCoordinates**: array of 3 triangle corners (vertices) the mouse is hitting in 3D model space coordinates
- **WorldSpaceTriangleCoordinates**: array of 3 triangle corners (vertices) the mouse is hitting in 3D world space coordinates.
- **NearestCoordinateIndex**: Index of nearest coordinate index of traced triangle, value of 0...2. Use the index to extract the coordinate from **ModelSpaceTriangleCoordinates** or **WorldSpaceTriangleCoordinates** array.

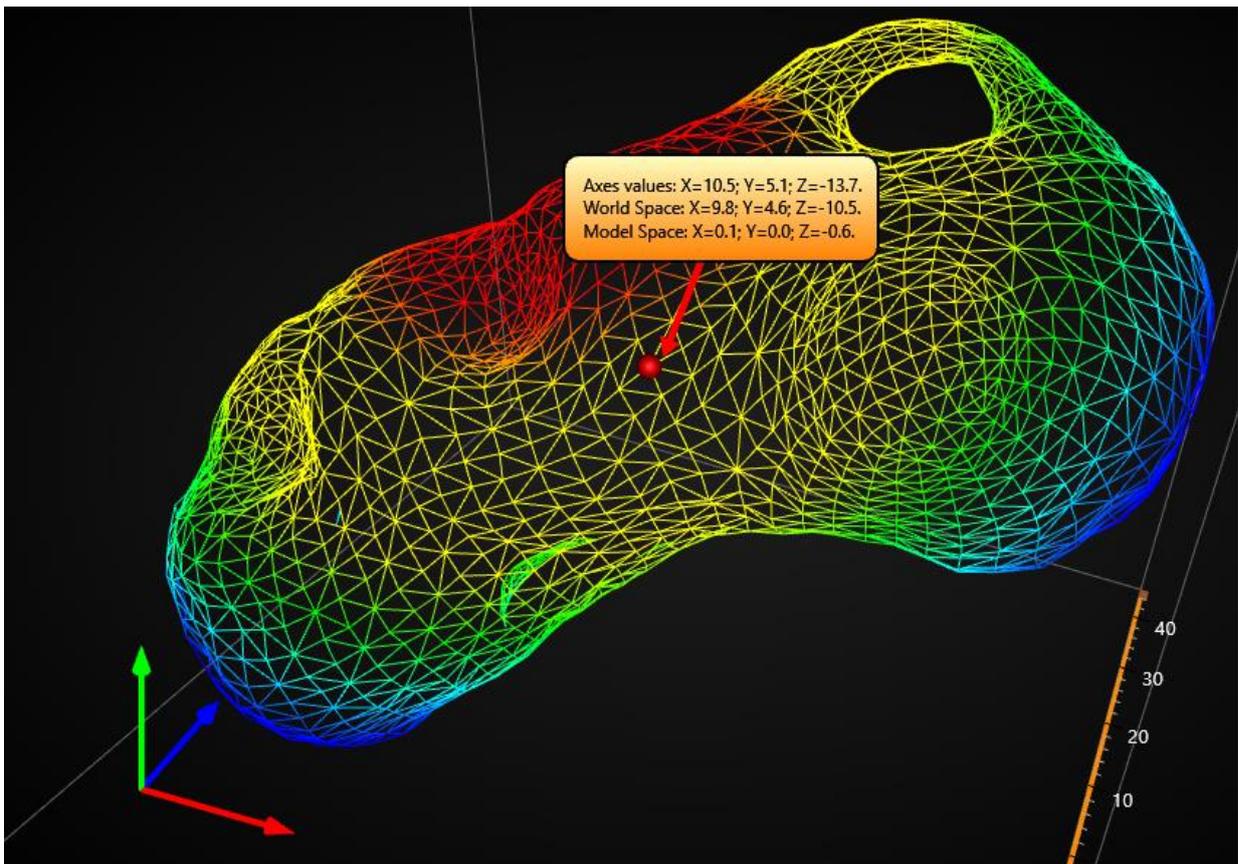


Figure 6-77. Tracing MeshModels with mouse. Traced result is shown in an Annotation.

6.15 VolumeModels

VolumeModels is a tool for volume data visualisation via Direct Volume Rendering. **VolumeModel** takes the volume data inside and visualizes it. LightningChart's volume rendering engine is based on the **Volume Ray Casting**.

An image is produced by the algorithm via the volume data sampling along the tracks of the rays which travel inside the dataset. A simple realization of hardware acceleration for **Volume Ray Casting** requires generating boundaries for a volume object. Usually, they are represented by a cube. High rendering quality without artefacts, and usage of the interchangeable ray function are the main advantages of this technology.

RayFunction is the core of the algorithm providing it with a very high level of flexibility. The technique is powerful because it specifies the way how the data is sampled and combined. This makes it a very useful tool for a feature extraction.

Note! VolumeModels are available only when DirectX 11 renderer is used.

6.15.1 Loading data

There are several ways how the data can be imported to the **VolumeModel**:

- Data can be supplied to **Data** property as a collection of images which represent slices of the dataset
- Data can be supplied directly to the constructor of the **VolumeModel** in various ways
- Data can be supplied to the **VolumeModel** via one of the load functions

Load functions and constructors allow supplying data as a collection of slices (similarly to Data property) or as a string with a path to the folder with the slices (as .Net supported image extension). The data can also be provided as a texture map created by our tool. A texture map consists of slices, but its supplement also needs an additional information about the number of slices on the picture. This is required for efficient usage of GPU input buffers. Texture maps can be created via ChartTools.CreateMap function. Direct input of texture map is used to speed up the start of an application for a very big dataset.

6.15.2 Properties

VolumeModel contains typical properties of a 3D object in LightningChart, for example **Visible**, **Rotation**, **Size**, **Position**, **MouseInteraction**, and **MouseHighLight**. In addition, the object has specific properties, which define how Volume Rendering engine handles it.

Brightness	
B	10
G	10
R	10
Darkness	
B	0.7
G	0.7
R	0.7
EmptySpaceSkipping	128
MouseHighlight	Simple
MouseInteraction	True
Opacity	0.15000000596046448
Position	
X	55
Y	50
Z	50
RayFunction	Accumulation
Rotation	
X	270
Y	0
Z	180
SamplingRateOptions	
Enabled	False
Inerthness	2
ManualSamplingRate	512
> SamplingRateRange	
TargetFPS	15
Size	
Depth	100
Height	75
Width	100
SliceRange	
▼ Max	
X	0.7
Y	0.8
Z	1
▼ Min	
X	0.3
Y	0.2
Z	0
Smoothness	2
Threshold	
▼ Max	
B	1
G	1
R	1
▼ Min	
B	0.5
G	0.5
R	0.5
Visible	True
XAxisBinding	Primary
YAxisBinding	Primary
ZAxisBinding	Primary

Figure 6-78. Property tree of VolumeModels

6.15.3 Ray Function

RayFunction property allows choosing one of the three ways of voxel sampling and composition available in LightningChart Volume Rendering Engine:

- **RayFunction.Accumulation** collects and combines as much data as possible. The visualization which is produced by this technique looks like a semi-transparent gel. The figure below shows an example of **RayFunction.Accumulation** application visualizing a medical dataset.



Figure 6-79. Example of a medical application for the **RayFunction.Accumulation**

- **RayFunction.MaximalIntensity** takes into account only the brightest values sampled by the ray. Visually it provides a very similar result to X-ray images. It allows to get an additional information about the internal structure of the object. **RayFunction.MaximalIntensity** applications for skeleton visualization and ultrasound wave's interference simulation are shown below.

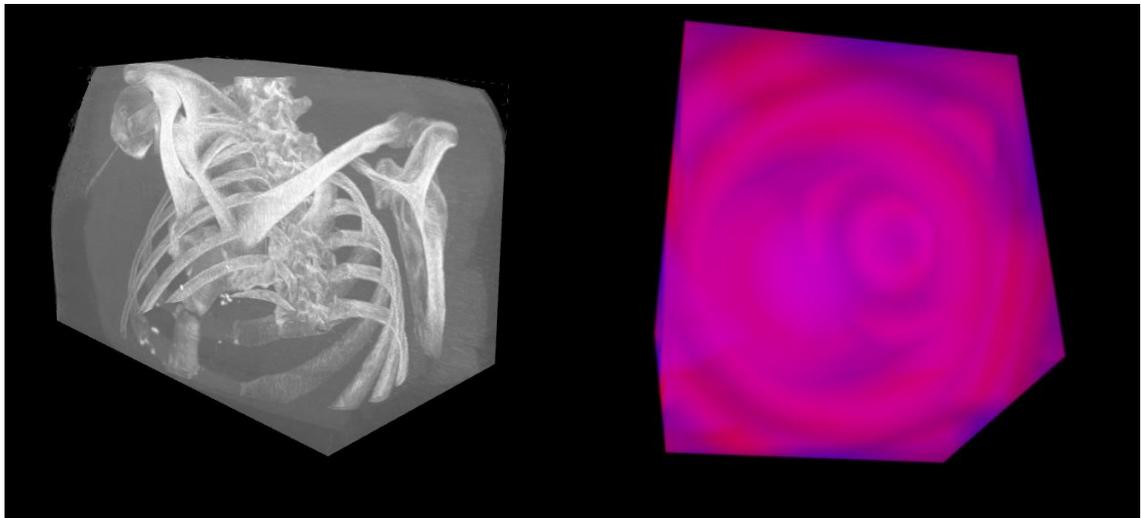


Figure 6-80. Examples of a Maximum Intensity Ray Function application

- **RayFunction.Isosurface** draws the model surface in a way that it looks like a polygonal model rendering. The result is very similar to those produced by **Indirect Volume Rendering**. Figures show examples of **RayFunction.Isosurface** applications for the visualization of human skull CT and simulation of water flow.

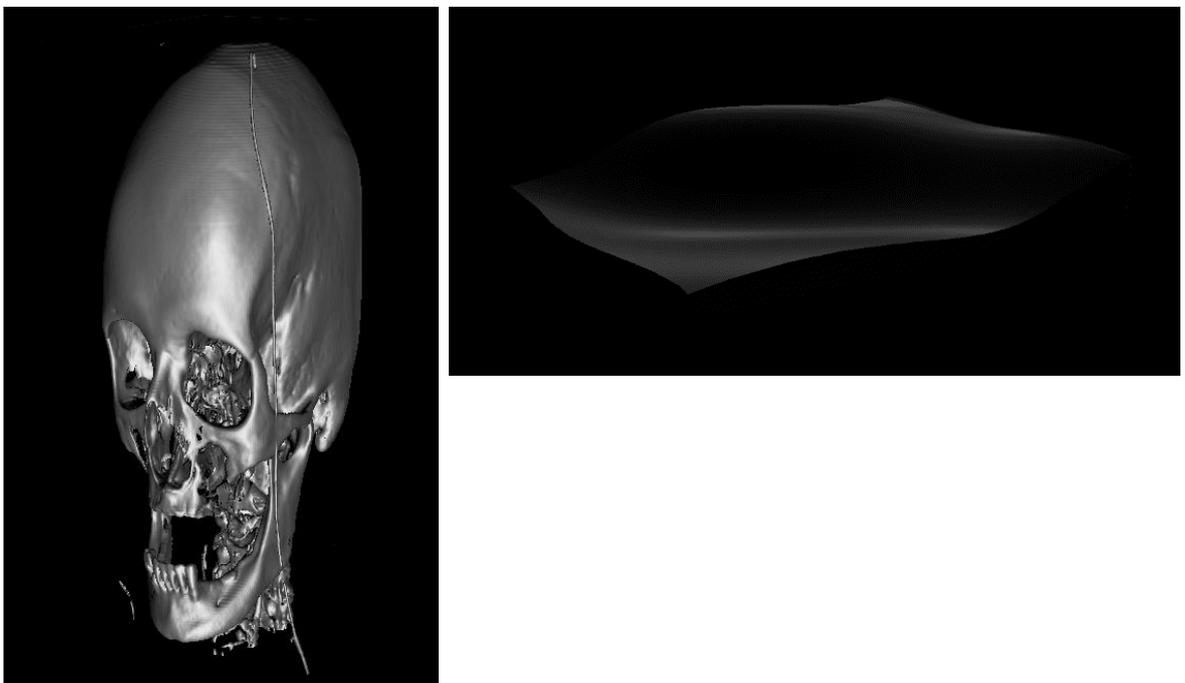


Figure 6-81. Examples of an Isosurface Ray Function application

6.15.4 Threshold

The Volume Rendering Engine can apply a threshold range by a property to the *VolumeModel*. There is a separate boundary for every colour channel. The voxel is visualized only if the corresponding color values are lower than the high boundary, and higher than the low boundary at all the channels. Acceptable areas are invisible. This property is not taken into consideration by the mouse hit test.

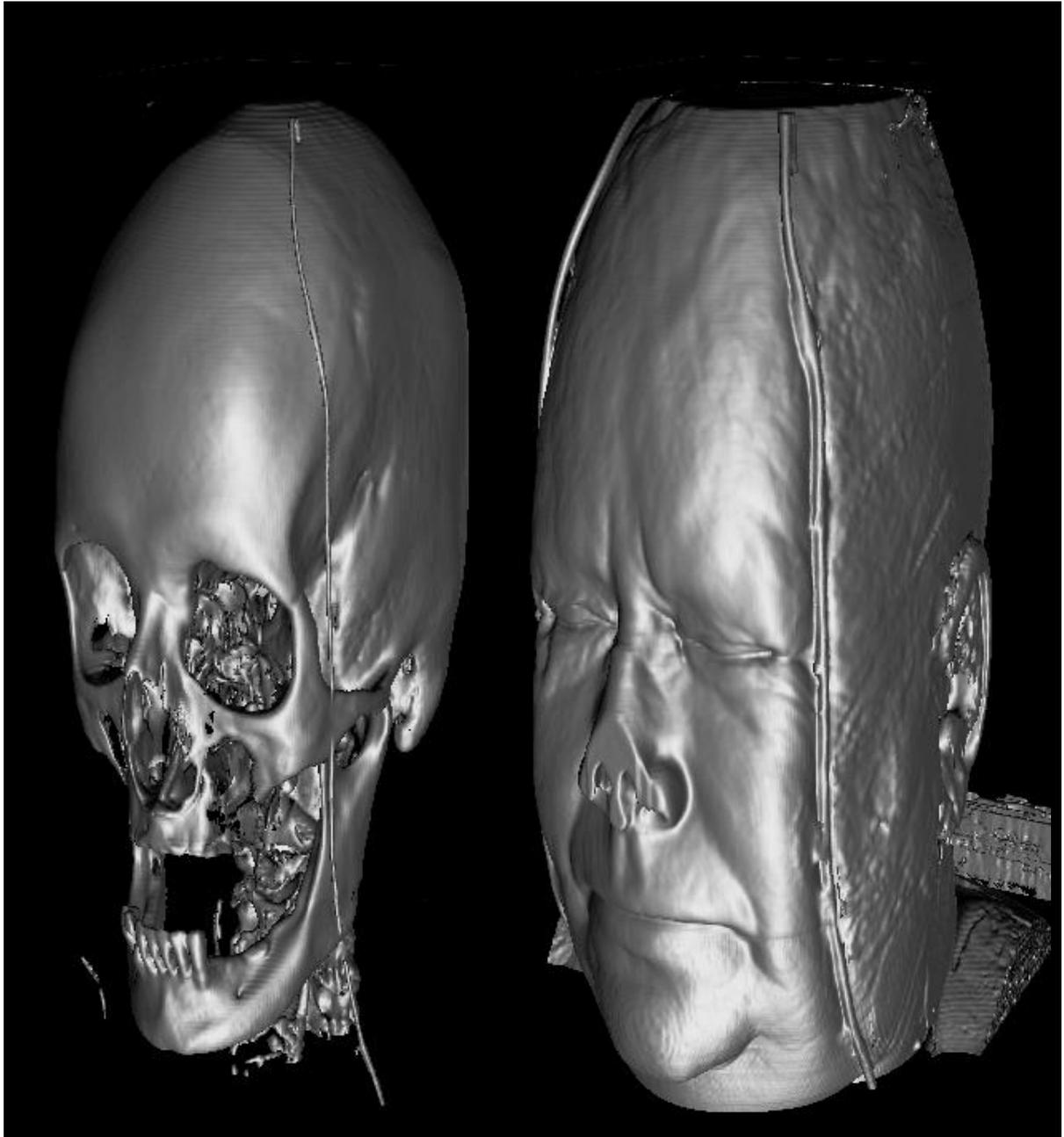


Figure 6-82. Example of two different threshold settings

6.15.5 Slice Range

SliceRange property allows cutting away a part of the ***VolumeModel***. It is a very useful tool for the exploration of the object's internal structure. ***SliceRange*** contains two boundaries, ***Min*** and ***Max***, both of which are represented by three pointing float values.

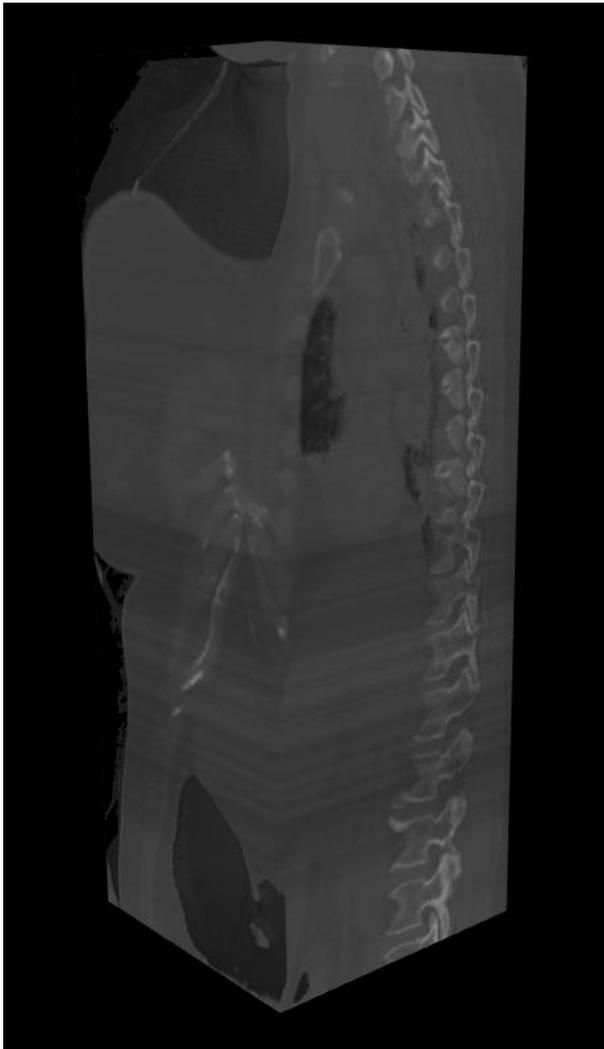


Figure 6-83. Example of Accumulation Ray Function and ***SliceRange*** modification

6.15.6 Sampling Rate Options

SamplingRate is a very important property to the final image quality. It defines how often the volume dataset is sampled along the ray's track. Higher **SamplingRate** produces better quality but requires more powerful hardware. **SamplingRate** influences **RayFunction** options, especially **Accumulation**. Artefacts produced by low sampling rate are less noticeable when using **Maximal Intensity**. Furthermore, **Isosurface** can be too sharp at a very high sampling rate. Usually, the sweet spot equals the number of voxels on the side which is placed along the ray tracks.

SamplingRateOptions contains several options for **SamplingRateManager**. **SamplingRateManager** is needed to reach the optimal balance between quality and frame rate for a hardware. By default, **SamplingRateManager** is turned on by the property **Enabled** being set **true**. If set **false**, **ManualSamplingRate** value will be used. **SamplingRateRange** defines the boundaries for **SamplingRateManager**. **Inertness** specifies how rapid is the reaction of sampling rate in case of performance changes. **TargetFPS** is a target value, which sampling rate manager tries to achieve.

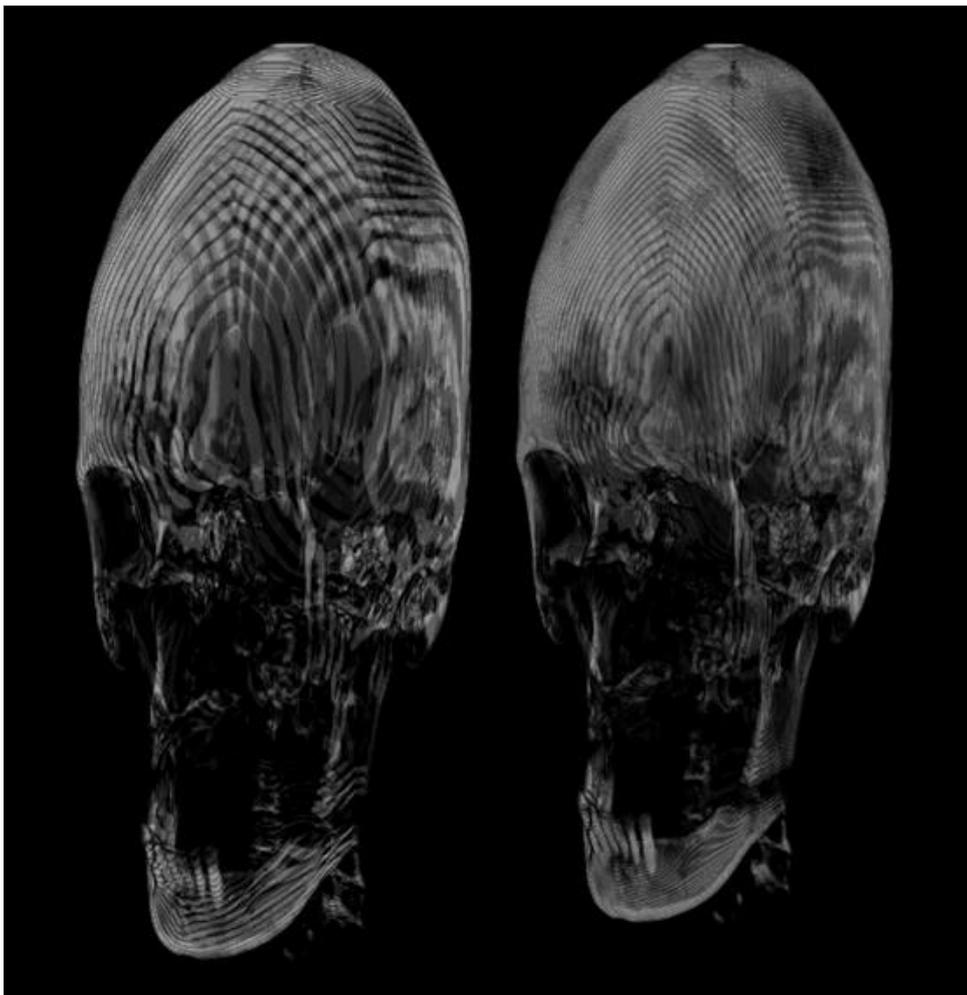


Figure 6-84. Example of low sampling rate: 32(left), 64(right)

6.15.7 Smoothness

Smoothness property prevents too high detalization of the surface. It smoothens the surface of the model and reduces some noise and other artefacts.

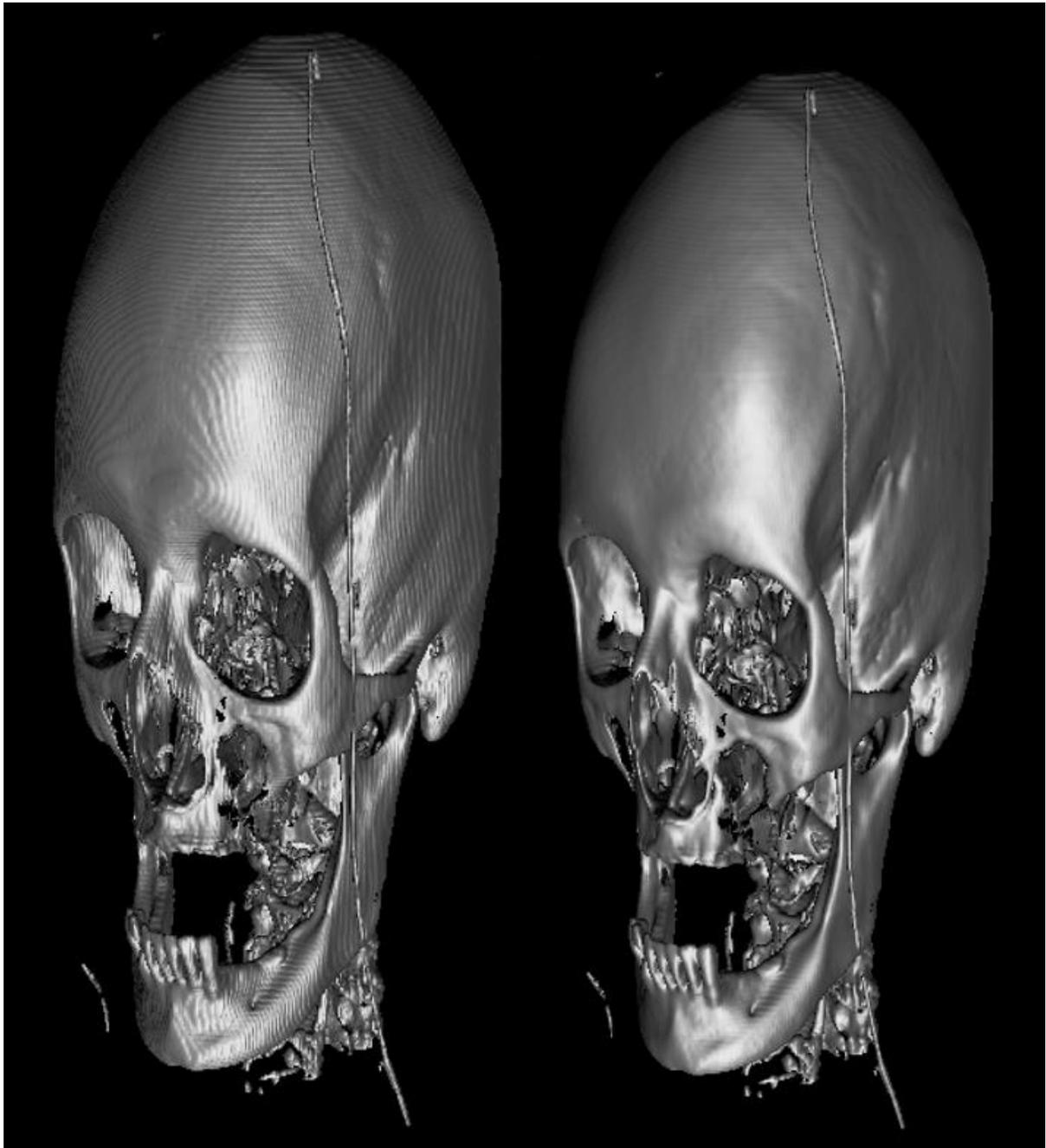


Figure 6-85. Example of too high sampling rate, fixed by smoothness property

6.15.8 EmptySpaceSkipping

EmptySpaceSkipping property defines a resolution of empty space, skipping sampling. A low value (16-32) of **EmptySpaceSkipping** improves the performance but can cause artefacts in the model edges.



Figure 6-86. Example of too low EmptySpaceSkipping property value

6.15.9 Opacity

Opacity specifies the behaviour of **Accumulation** option of **RayFunction**. The lower the **Opacity**, the more transparent the object will be.



Figure 6-87. Example of Accumulation Ray Function Opacity modification: 15% (left), 45(right)

6.15.10 Brightness and Darkness

These properties define the image's transfer function. Every change has its own transfer function. It is represented by the linear function: $output = Brightness * input - Darkness$

6.16 Rectangle3D objects

Rectangle3D allows presenting a rectangle, turned to any angle, at any size, at any location. They can be added to **View3D.Rectangles** list. Rectangles can also act as planes by defining their size according to **View3D.Dimensions**.

Set **Size** in 3D world dimensions (not X, Y or Z axis values) as **Width** and **Height**. Set the center point via **Center** property, defined in X, Y and Z axis values. **Rotation** property specifies the rotation in degrees.

Fill settings can be modified via **Fill** property. Solid color and bitmap fills are available. To use bitmap fill, set the bitmap in **Image**, and enable **UseImage**. When setting **Fill.Layout = Stretch**, the bitmap stretches to fill the rectangle. By setting **Fill.Layout = Tile**, the same bitmap is tiled to fill the rectangle. The tile count can be altered via **Fill.TileCountWidth** and **Fill.TileCountHeight** properties.

▼	Misc	
▼	Center	
	X	50
	Y	5
	Z	50
▼	Fill	
	> Image	 System.Drawing.Bitmap
	ImageAlpha	255
	Layout	Stretch
	▼ Material	
	AmbientColor	 Black
	DiffuseColor	 150, 50, 50, 50
	EmissiveColor	 Black
	SpecularColor	 Gray
	SpecularPower	5
	TileCountHeight	10
	TileCountWidth	10
	UseImage	True
	MouseHighlight	Blink
	MouseInteraction	True
▼	Rotation	
	X	0
	Y	0
	Z	0
▼	Size	
	Height	100
	Width	100
	Visible	True
	XAxisBinding	Primary
	YAxisBinding	Primary
	ZAxisBinding	Primary

Figure 6-88. Properties of Rectangle3D objects.

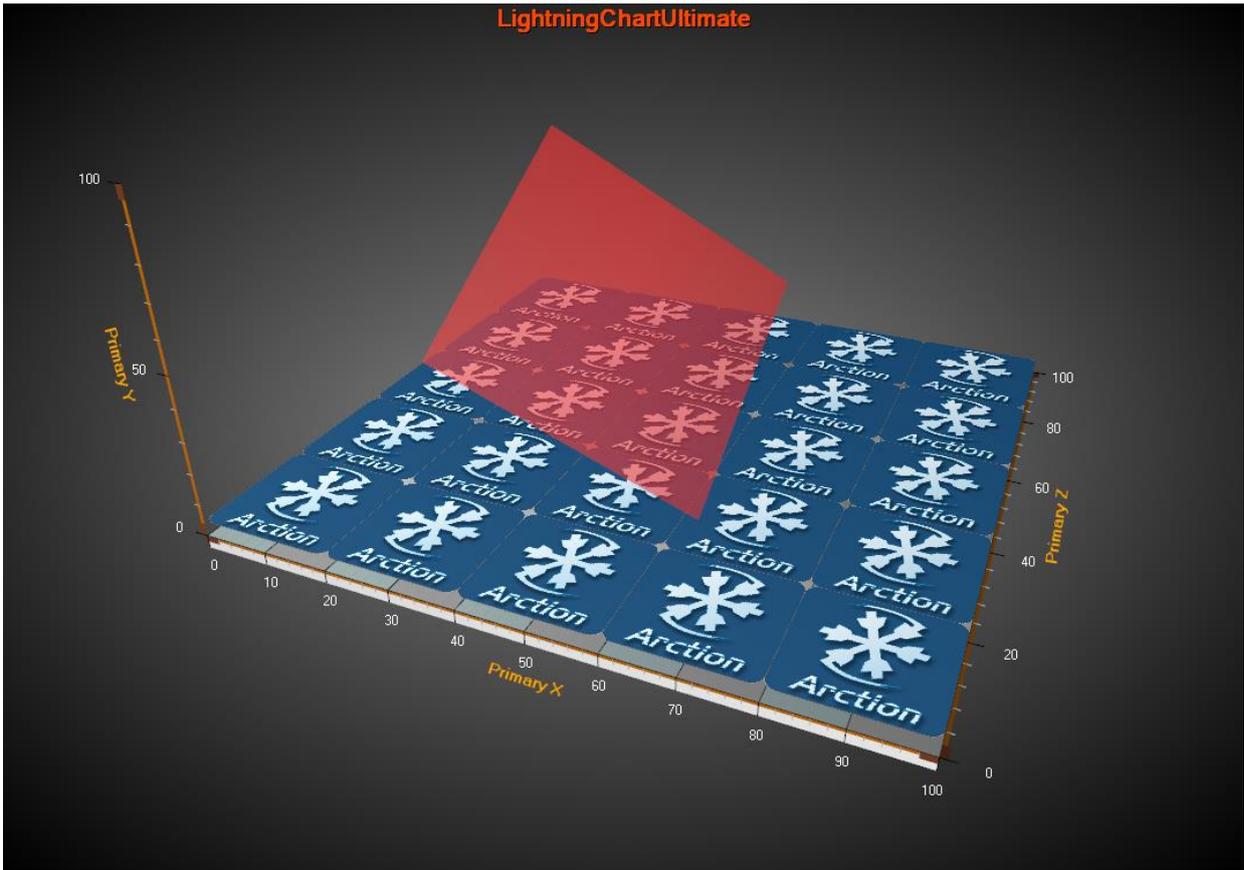


Figure 6-89. Two Rectangle3D objects in View3D. The bottom blue one shows a bitmap fill with Layout = Tile. The red rotated rectangle on the top is configured with translucent color.

6.17 Polygon3D objects

Polygon3D objects allow presenting a 2D polygon stretched to given Y range. They can be added to **View3D.Polygons** list.

Define the polygon path in X and Z axis values and store it in **Points** array. Set the Y range with **YMin** and **YMax** values.

Material.Diffuse controls the main color of the rectangle. Rotate the polygon to another angle with **Rotation.X**, **Rotation.Y** and **Rotation.Z** in degrees.

▼ Misc	
▼ Material	
AmbientColor	Black
DiffuseColor	Magenta
EmissiveColor	Black
SpecularColor	Gray
SpecularPower	5
MouseHighlight	Simple
MouseInteraction	True
> Points	Polygon3DPoint[] Array
▼ Rotation	
X	0
Y	0
Z	0
Visible	True
XAxisBinding	Primary
YAxisBinding	Primary
YMax	20
YMin	0
ZAxisBinding	Primary

Figure 6-90. Properties of Polygon3D objects.

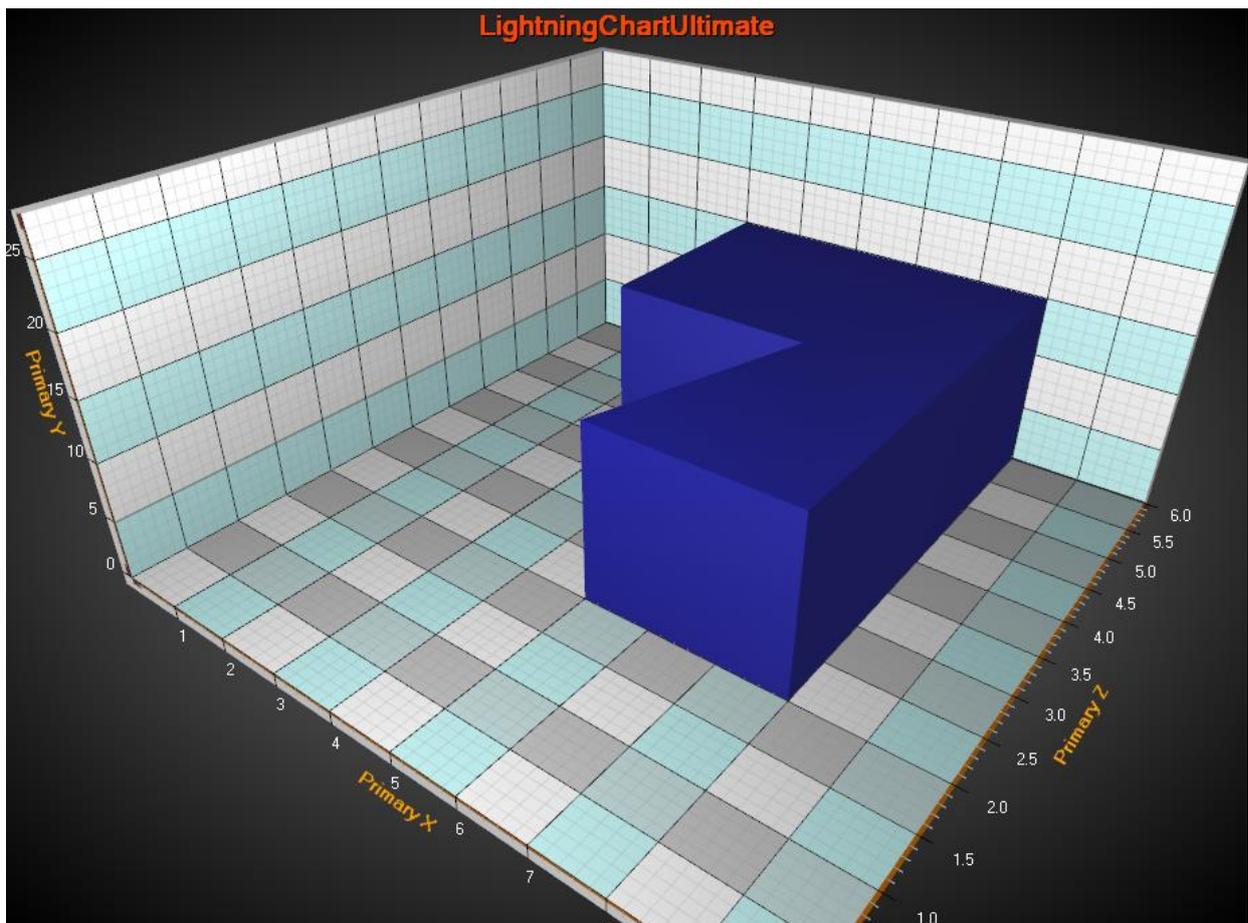


Figure 6-91. A 6-point polygon ranging from YMin = 0, YMax = 15.

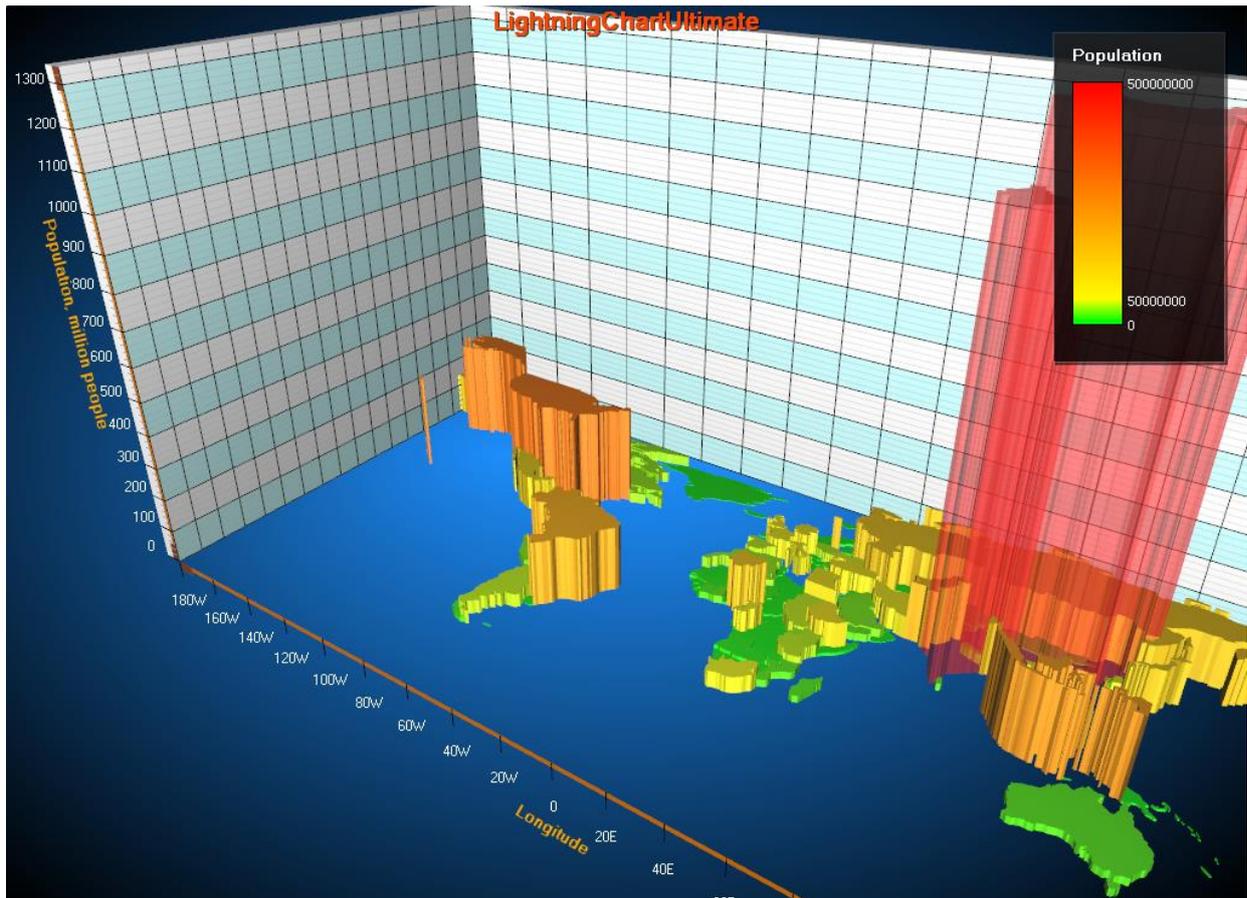


Figure 6-92. World population shown with Polygon3D objects. A Polygon3D object is drawn based on each region of map data. The amount of population of a country is used to color the polygon and to set its YMax. China and India are shown with translucent colors because of their high population.

6.18 Zooming, panning and rotating

ZoomPanOptions properties can be used to control the zooming, panning and rotation settings.

ZoomPanOptions	
AutoFit	False
AxisMouseWheelAction	Pan
BoxZoomingOutCrossVisible	True
BoxZoomOutFactor	2
LeftDoubleClickAction	Zoom ToFit
LeftMouseButtonAction	Rotate
LimitBoxZoomInsideGraph	True
MiddleMouseButtonAction	Pan
MouseWheelZoomEnabled	True
MouseWheelZoomFactor	1,1
MultiTouchPanEnabled	True
MultiTouchZoomEnabled	True
PanSensitivity	1
> RectangleZoomingThreshold	
RightMouseButtonAction	Pan
RightToLeftZoomAction	ZoomOut
RotationSensitivity	1
WheelAreaThickness	2
ZoomBoxColor	 30, 255, 165, 0
> ZoomInBoxLineStyle	
> ZoomOutBoxLineStyle	

None
Pan
Rotate
PanPrimaryXZ
PanPrimaryXY
PanPrimaryYZ
ZoomXY
ZoomXZ
ZoomYZ
ZoomX
ZoomY
ZoomZ

Figure 6-93. ZoomPanOptions properties and sub-properties, with LeftMouseButton / MiddleMouseButtonAction / RightMouseButtonAction options on the right.

Depending on the settings, zooming can be performed with mouse wheel, by touch screen pinching/spreading, or by painting a box on selected 3D plane. Panning, box zooming, and rotating can all be performed by left, middle or right mouse button, as they are configurable. Panning can be made for the whole 3D chart, or so that primary axes are adjusted while keeping the 3D scene location the same.

6.18.1 Mouse wheel zooming

Scroll mouse wheel up to zoom in, and down to zoom out. Use **MouseWheelZoomFactor** to adjust the amount of applied zoom with every mouse wheel event. To disable mouse wheel zooming, set **MouseWheelZoomEnabled** to **false**. By default, this is set **true**.

6.18.2 Box zooming

To enable box zooming, assign the box zooming to a mouse button action property. For example, **LeftMouseButtonAction = ZoomXZ** causes the box zoom to apply to XZ plane. Y dimension is not affected. Set **ZoomXZ** or **ZoomYZ** respectively for zooming other planes.

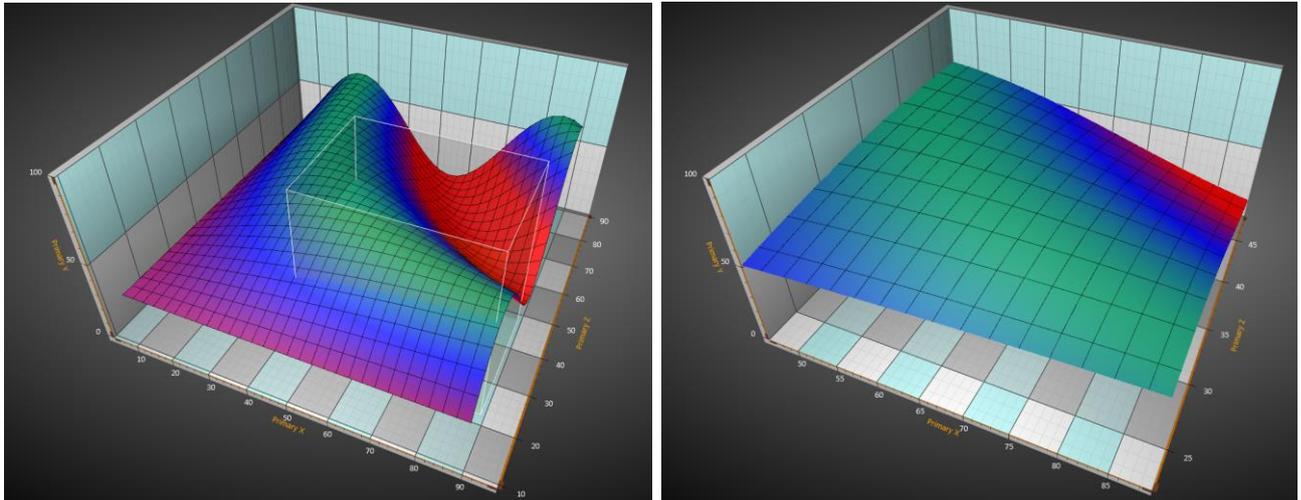


Figure 6-94. On the left, XZ-plane box zooming in progress. On the right, the outcome of the zooming. X and Z axis ranges are modified while Y axis range stays the same.

Zoom in by dragging box from left to right. The zoomed ranges are applied to axes related to the selected plane.

To zoom only specific dimension, X, Y or Z, select **ZoomX**, **ZoomY** or **ZoomZ**.

Zoom out by dragging box from the right to left. Zooming out is applied by factor set in **BoxZoomOutFactor**. Zooming out shows a cross in the front of the box. It can be disabled by setting **BoxZoomingOutCrossVisible = False**.

6.18.3 ZoomPadding

ZoomPadding property defines the amount of empty space left between the 3D model and the margins after a zooming operation. **ZoomPadding** has no effect when moving the chart or zooming manually, for example by mouse scrolling. Furthermore, **ZoomPadding** does not apply to rectangle-based zooming.

Setting **ZoomPadding** in WinForms:

```
chart.View3D.ZoomPanOptions.ZoomPadding = new Padding(10, 30, 10, 10);
```

Setting **ZoomPadding** in Wpf:

```
chart.View3D.ZoomPanOptions.ZoomPadding = new Thickness(10, 30, 10, 10);
```

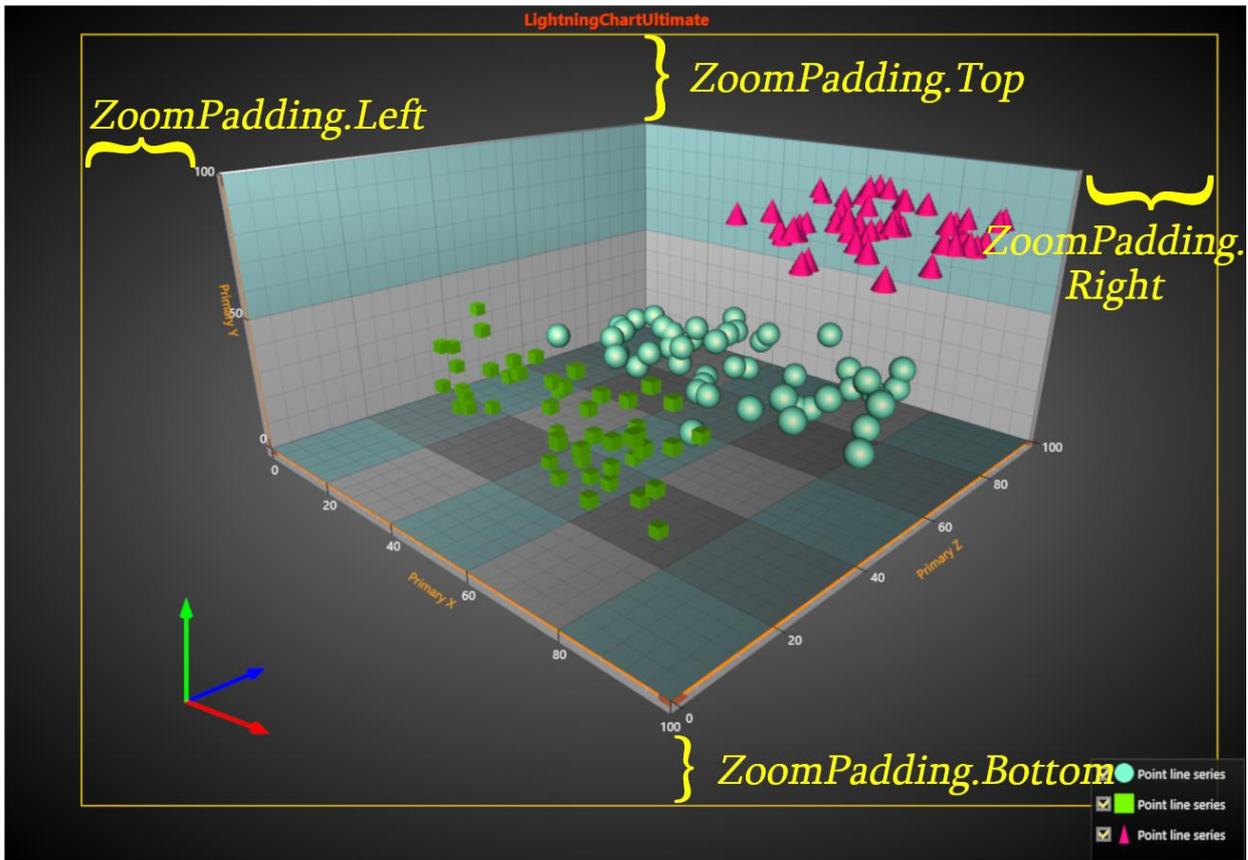


Figure 6-95. **ZoomPadding** leaves empty space between data/labels and the margins if for example **ZoomToDataAndLabels** operation is used as in this case.

6.18.4 **ZoomToDataAndLabels**

In **View3D**, **ZoomToDataAndLabels** operation causes the available area, limited by margins and **ZoomPadding**, to be used as optimally as possible by moving the camera closer/farther. Axes, labels, series data and markers are all kept visible. Chart title, annotations and legend boxes are ignored as their position is defined in screen coordinates. **ZoomToDataAndLabels** maintains the viewing angle while the contents of the view are centered.

By default, **LeftDoubleClickAction** property is set as **ZoomToDataAndLabels**, meaning double-clicking the mouse left button activates the operation. Disable this by changing the property to **Off**. In code, **ZoomToDataAndLabels** can be invoked by **View3D.ZoomToFit(ZoomArea3D.DataAndLabelsArea)** method.

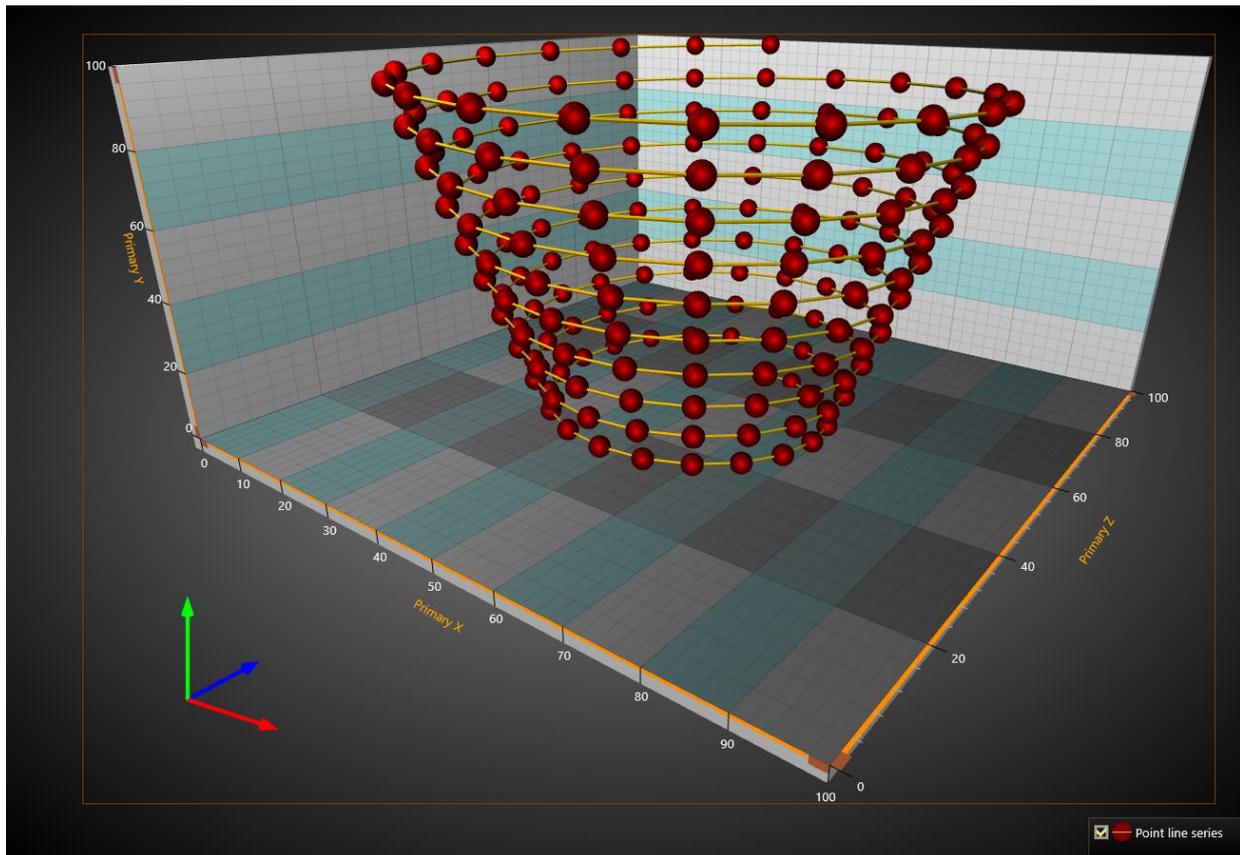


Figure 6-96. **ZoomToDataAndLabels** operation has been activated. Data series, axes, walls etc. have been optimally fitted within the margins. Orientation arrows follow the bottom-left corner of the graph area but legend boxes are ignored. **ZoomPadding = 0** for all edges, thus no empty space is left between the margins and the **DataAndLabelsArea**.

Note that **ZoomToDataAndLabels** takes **MinimumViewDistance** property of the camera to account, meaning that in some cases the full available graph area might not be used as the camera can't get close enough to the chart. A **ChartMessage** notification is sent in this case.

6.18.5 Rotating and panning

Camera can be rotated around the 3D model by pressing the assigned mouse button down and by dragging horizontally or vertically. **RotationX**, **RotationY** and **RotationZ** properties are updated.

When a mouse button action is set to **Pan**, panning updates **Target** property of **Camera**. When mouse button action is set to **PanPrimaryXZ**, **PanPrimaryXY** or **PanPrimaryYZ**, the primary X, Y and Z axes ranges are adjusted. For example, **PanPrimaryXZ** adjusts X and Z axes when dragging with mouse. Secondary X, Y and Z axes are not altered.

Set **LeftMouseButtonAction** / **MiddleMouseButtonAction** / **RightMouseButtonAction** to **Pan/PanPrimaryXZ/PanPrimaryXY/PanPrimaryZ** to enable panning. Set it to **Rotate** to enable rotating. To disable panning and rotating from left mouse button, set it to **None**.

Use **PanSensitivity** to control the amount of applied panning. Respectively, use **RotationSensitivity** to control the amount of applied rotation.

6.18.6 Zooming with touch screen

Set two fingers on the chart, and pinch them closer to zoom out, or away to zoom in. To disable zooming with touch screen, set **MultiTouchZoomEnabled** to **False**.

6.18.7 Panning with touch screen

Set two fingers on the chart and move them to the same direction to apply panning. To disable panning with touch screen, set **MultiTouchPanEnabled** to **False**.

6.18.8 Using mouse wheel over an axis

When mouse wheel is scrolled over an axis, the chart makes axis-specific zooming or panning. **WheelAreaThickness** adjusts how wide the mouse wheel sensitive area is near the axis. **AxisMouseWheelAction** can be used to select between zooming and panning.

6.18.9 Zooming, rotating and panning by code

3D view is rotated by moving the **View3D.Camera** with **RotationX**, **RotationY** and **RotationZ** properties. With **Perspective** and **Orthographic** camera, zoom can be done by setting **ViewDistance**. With **OrthographicLegacy** camera, the **Dimensions** are changed to achieve zooming. Panning is done by setting camera **Target** as 3D model coordinates.

6.19 Legend boxes

Legend boxes in View3D are largely similar to ViewXY's legend boxes (see chapter 5.21). However, only one legend box is allowed per graph. Also, segment-based properties do not exist, since axes in View3D cannot be divided in segments. Modify the legend box properties via **View3D.LegendBox**.

LegendBox	
AllowMouseResize	True
AutoSize	True
BorderColor	<input type="color" value="#402555"/> 40, 255, 255, 255
BorderWidth	1
Categorization	None
CategoryColor	<input type="color" value="white"/> White
CategoryFont	Segoe UI, 10pt, style=Bold
CheckBoxColor	<input type="color" value="#140255"/> 140, 255, 255, 255
CheckBoxSize	15
CheckMarkColor	<input type="color" value="khaki"/> Khaki
Fill	
Height	108
HighlightSeriesOnTitle	True
HighlightSeriesTitleColor	<input type="color" value="yellow"/> Yellow
Layout	VerticalColumnSpan
MouseHighlight	Simple
MouseInteraction	True
MoveByMouse	True
MoveFromSeriesTitle	True
Offset	
Position	BottomRight
ScrollBarVisibility	Both
SeriesTitleColor	<input type="color" value="white"/> White
SeriesTitleFont	Segoe UI, 10pt
Shadow	
ShowCheckboxes	True
ShowIcons	True
SurfaceScales	
UnitsColor	<input type="color" value="white"/> White
UnitsFont	Segoe UI, 9pt
UseSeriesTitlesColors	False
ValueLabelColor	<input type="color" value="white"/> White
ValueLabelFont	Segoe UI, 9pt
Visible	True
Width	161

Figure 6-97. Legend box properties in View3D

6.19.1 Hiding surface series palette scales

View3D has **SurfaceScales** property instead of ViewXY's **IntensityScales**. To hide the palette scale in a legend box, set **SurfaceScales.Visible** = **False**. To resize it, set **ScaleSizeDim1** and **ScaleSizeDim2** properties.

6.19.2 Positioning legend boxes in View3D

As in ViewXY, View3D's legend boxes can be placed automatically or manually. Automatic placement allows them to be aligned to the left/top/right/bottom side of the view, or the graph area. Control the position with **Position** property. Some positioning options take margins into account while some do not.

Options ignoring margins (placing the legend box in the margin area):

TopCenter, TopLeft, TopRight, LeftCenter, RightCenter, BottomLeft, BottomCenter, BottomRight, Manual

Options placing the legend box inside the margin area:

GraphTopCenter, GraphTopLeft, GraphTopRight, GraphLeftCenter, GraphRightCenter, GraphBottomLeft, GraphBottomCenter, GraphBottomRight

Offset property shifts the position by given amount *from the position determined by Position* property.

```
// Setting legend box position, offset shifts from RightCenter position
chart.View3D.LegendBox.Position = LegendBoxPosition.RightCenter;
chart.View3D.LegendBox.Offset = new PointIntXY(-15, -70);
```

Manual positioning calculates the offset from the top-left corner of the legend box to the view's top-left corner. Note that this differs from **TopLeft** option, which is calculated from the top of the graph area.

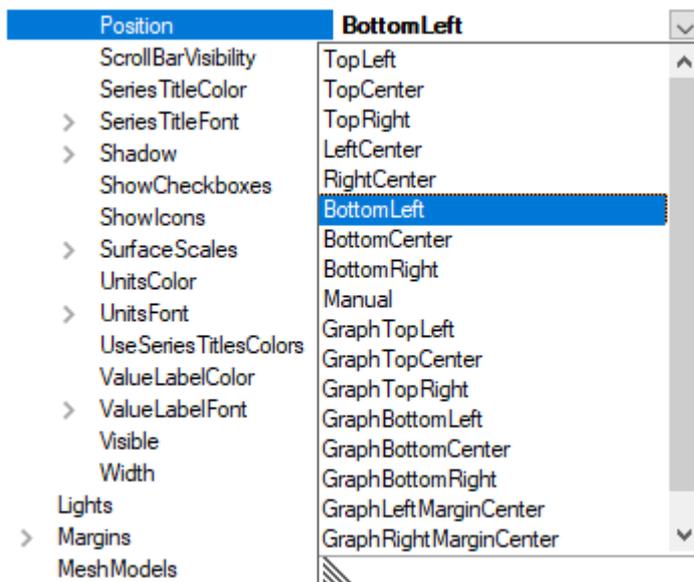


Figure 6-98. Positioning options for legend box. Graph.. options place the legend box inside the margins.

6.20 Clipping objects within axis ranges

By setting **ClipContents** property to **True**, series, rectangles and mesh models are clipped inside axis value ranges. The axes are always stretched for a dimension, so when clipping is enabled, it prevents rendering outside the walls.

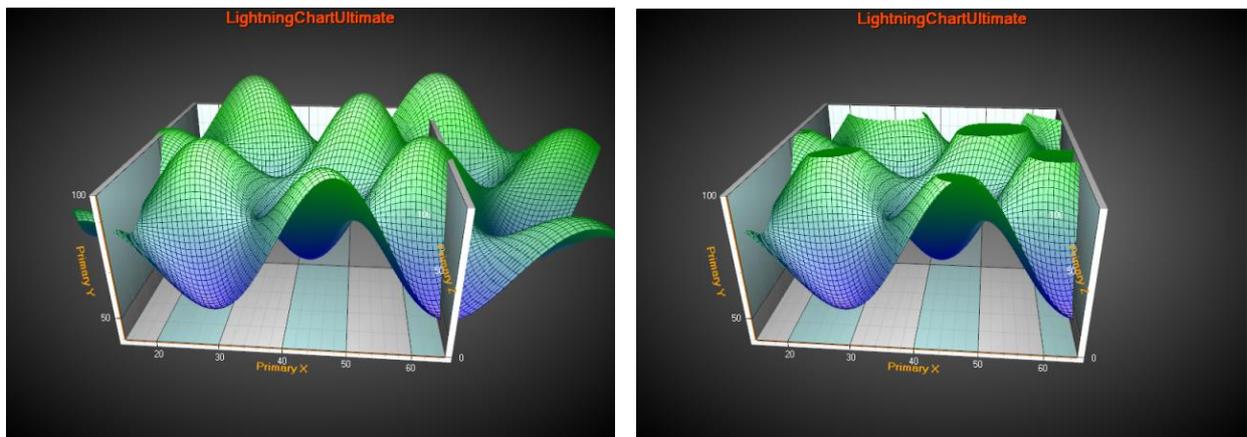


Figure 6-99. On the left, **ClipContents** is not used. Series render outside axis ranges. On the right, **ClipContents** is enabled.

Note that clipping does not modify the series data set itself. Clipping occurs only in rendering stage. Also, mouse hit test will take effect also outside the walls for invisible, clipped objects.

When clipping is enabled, all lines in the chart are automatically set to line width of 1.

6.21 Annotation3D

Annotation3D collection allows adding annotations into the 3D scene. In general, they are similar to ViewXY's **Annotations** (see chapter 5.20), with the exception of **Target** and **Location** properties using X, Y and Z dimensions.

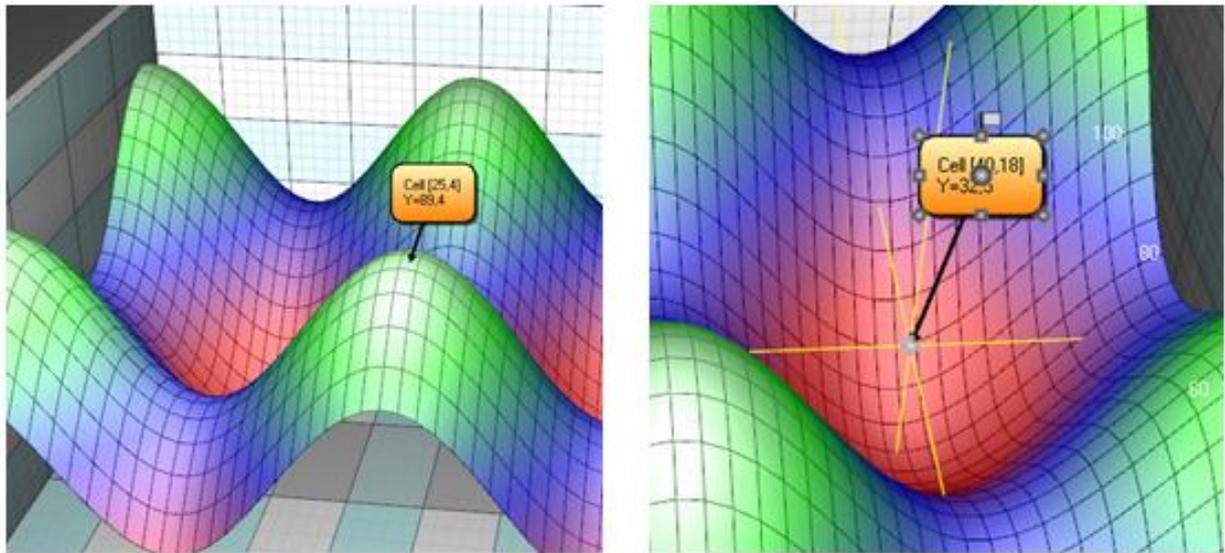


Figure 6-100. Annotation3D object displaying the value of a 3D series. Crosshair cursor can be used to aid the target movement.

Target can be moved by mouse in 3D. For aiding the movement, annotation shows cross-hair lines when mouse is over the **Target** node. Set **ShowTargetCrosshair** property to **Auto/On/Off** and adjust the line style in **TargetCrosshairLineStyle**.

7. Coordinate system converters

The following coordinate system converters are available in **CoordinateConverters** namespace, which complements View3D usage.

- Cartesian 3D <-> Spherical 3D
- Cartesian 3D <-> Cylindrical 3D

7.1 SphericalCartesian3D

SphericalCartesian3D converter class converts between spherical and 3D cartesian coordinates.

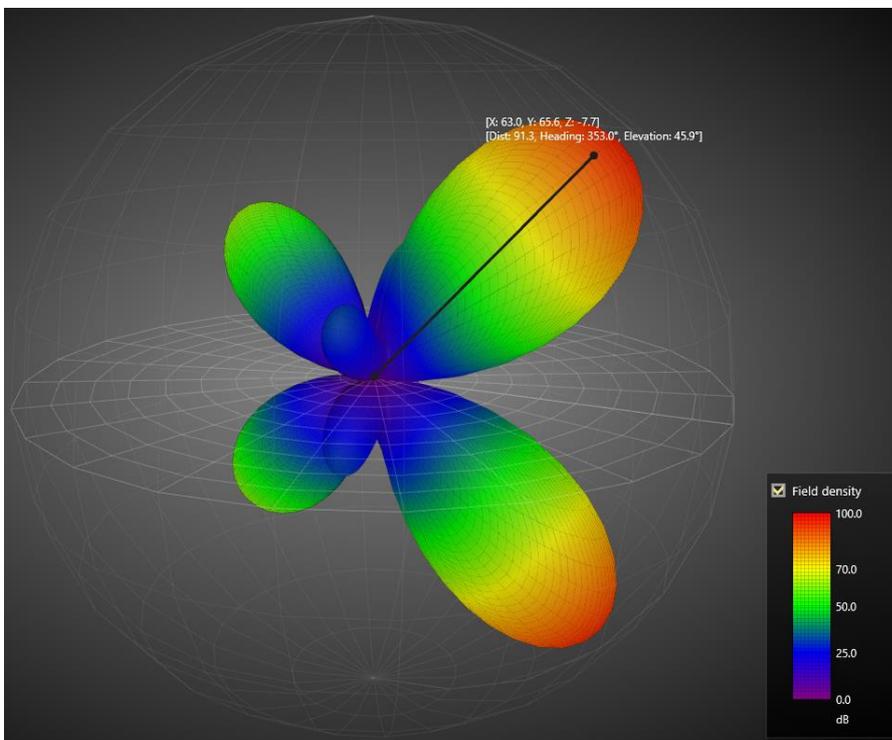


Figure 7-1. Example created with **SphericalCartesian3D** converter. Data points of **SurfaceMeshSeries3D** and the grid are defined in spherical coordinates. Annotation tracks the nearest data point and displays its value in spherical coordinates.

Spherical data points are defined by **SphericalPoint** objects which contain the following fields:

- **Distance**: Distance from origin (0,0,0)
- **ElevationAngle**: Elevation angle. Also called as Elevation or Altitude, measured from XZ plane. ElevationAngle is 90 degrees - Inclination angle.
- **HeadingAngle**: Heading angle. Also called as azimuth and absolute bearing

For elevation, the XZ plane is the reference plane. (e.g. equatorial plane). Elevation is an angle measured from that plane.

Note! This converter class expects the View3D.Dimensions to be equal (cubic), otherwise the conversion result may have to be scaled by user-side code.

View3D's series typically take the data input as X, Y, Z values. These values can be found e.g. in **SeriesPoint3D**, **SurfacePoint3D** and **PointDouble3D** objects.

7.1.1 Converting from spherical to cartesian

To convert a **SphericalPoint** to cartesian coordinate, use **SphericalCartesian3D.ToCartesian()** method. It accepts data input as

- SphericalPoint point
- SphericalPoint[] array
- SphericalPoint[,] matrix

Alternatively, convert by using **ToCartesian()** extension method for spherical points.

```
// Create spherical points matrix
SphericalPoint[,] sphericalData = CreateSurfaceData();

// Convert matrix to cartesian matrix
SurfacePoint[,] xyzData = sphericalData.ToCartesian();
```

Converting matrix to cartesian in Bindable WPF chart:

```
SurfacePointMatrix xyzData = sphericalData.ToCartesian();
```

7.1.2 Converting from cartesian to spherical

To convert a cartesian point to spherical point, use **SphericalCartesian3D.ToSpherical()** method. It accepts data input as **PointDouble3D** point with X, Y and Z fields.

Alternatively, convert a point by using **ToSpherical()** extension method.

```
// Define cartesian point
PointDouble3D point = new PointDouble3D(50, 20, 40);

// Convert to spherical point
SphericalPoint sp = point.ToSpherical();
```

7.2 CylindricalCartesian3D

Converter class to convert between cylindrical and 3D cartesian coordinates.

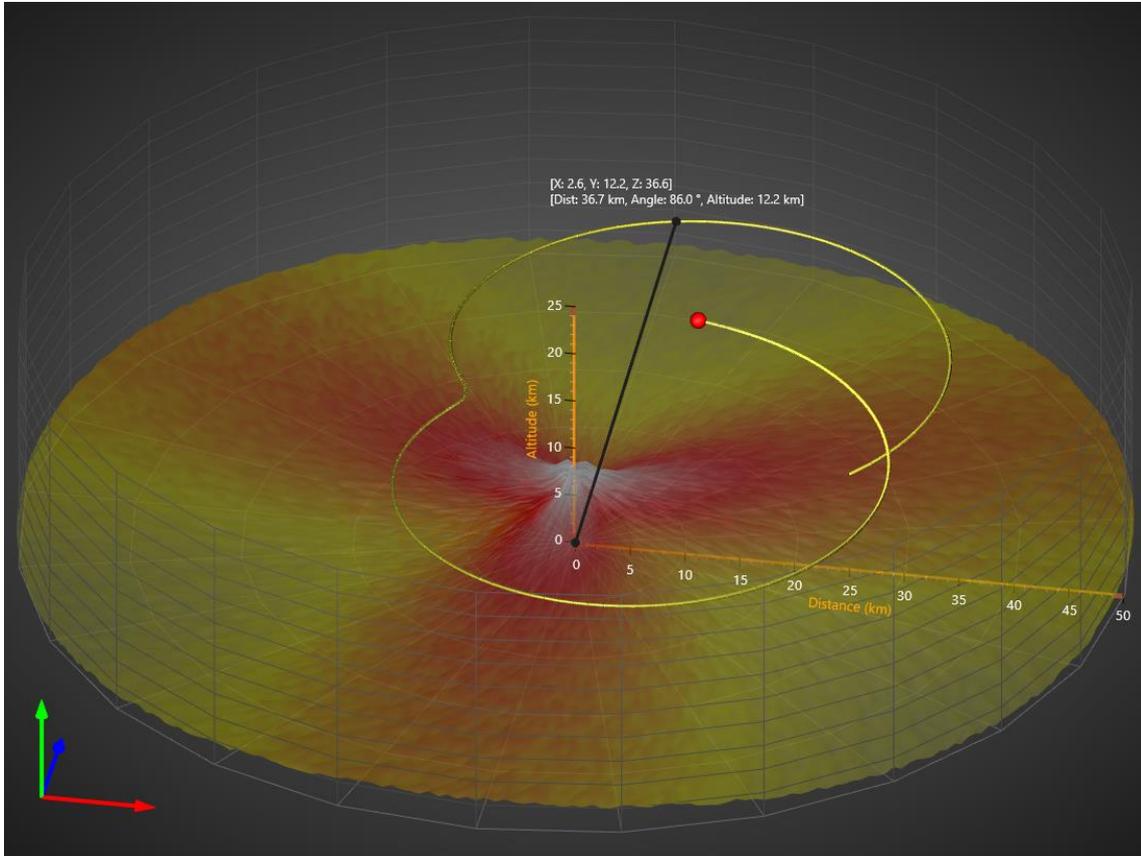


Figure 7-2. Example created with `CylindricalCartesian3D` converter. Data points of `SurfaceMeshSeries3D` and the grid are defined in cylindrical coordinates. Annotation tracks the nearest data point of a `PointLineSeries3D` and displays its value in cylindrical coordinates.

Cylindrical points are defined by `CylindricalPoint` objects, which contain the following fields:

- **Distance:** Distance along XZ plane
- **Y:** Y value
- **Angle:** Heading angle, also called as azimuth and absolute bearing

Note! This converter class expects `View3D.Dimensions.X` and `View3D.Dimensions.Z` to be equal, otherwise the conversion result regarding **Angle** and **Distance** (or X and Z) may have to be scaled by user-side code.

`View3D`'s series typically take the data input as X, Y, Z values. These values can be found e.g. in `SeriesPoint3D`, `SurfacePoint3D` and `PointDouble3D` objects.

7.2.1 Converting from cylindrical to cartesian

To convert a **CylindricalPoint** to cartesian coordinate, use **CylindricalCartesian3D.ToCartesian()** method. It accepts data input as

- CylindricalPoint point
- CylindricalPoint[] array
- CylindricalPoint[,] matrix

Alternatively, convert by using **ToCartesian()** extension method for cylindrical points.

```
// Create spherical points matrix
CylindricalPoint[,] cylindricalData = CreateData();

// Convert matrix to cartesian matrix
SurfacePoint[,] xyzData = cylindricalData.ToCartesian();
```

Converting matrix to cartesian in Bindable WPF chart:

```
SurfacePointMatrix xyzData = cylindricalData.ToCartesian();
```

7.2.2 Converting from cartesian to cylindrical

To convert a cartesian point to cylindrical point, use **CylindricalCartesian3D.ToCylindrical()** method. It accepts data input as **PointDouble3D** point with X, Y and Z fields.

Alternatively, a point can be converted by using **ToCylindrical()** extension method.

```
// Define cartesian point
PointDouble3D point = new PointDouble3D(50, 20, 40);

// Convert to spherical point
CylindricalPoint sp = point.ToCylindrical();
```

8. ViewPie3D

ViewPie3D presents data as pie and donut charts, in 3D.

ViewPie3D	3D pie/donut view
Annotations	(Collection)
AutoSizeMargins	False
> Border	Border
> Camera	
DonutInnerPercents	50
ExplodePercents	10
> LegendBox3DPie	LegendBoxPie3D
LightingScheme	DirectionalFromCamera
Lights	(Collection)
> Margins	30, 30, 30, 30
> Material	
Rounding	40
StartAngle	0
Style	Pie
Thickness	25
TitlesNumberFormat	0 USD
TitlesStyle	Values
Values	(Collection)
> ZoomPanOptions	

Figure 8-1. ViewPie3D object tree.

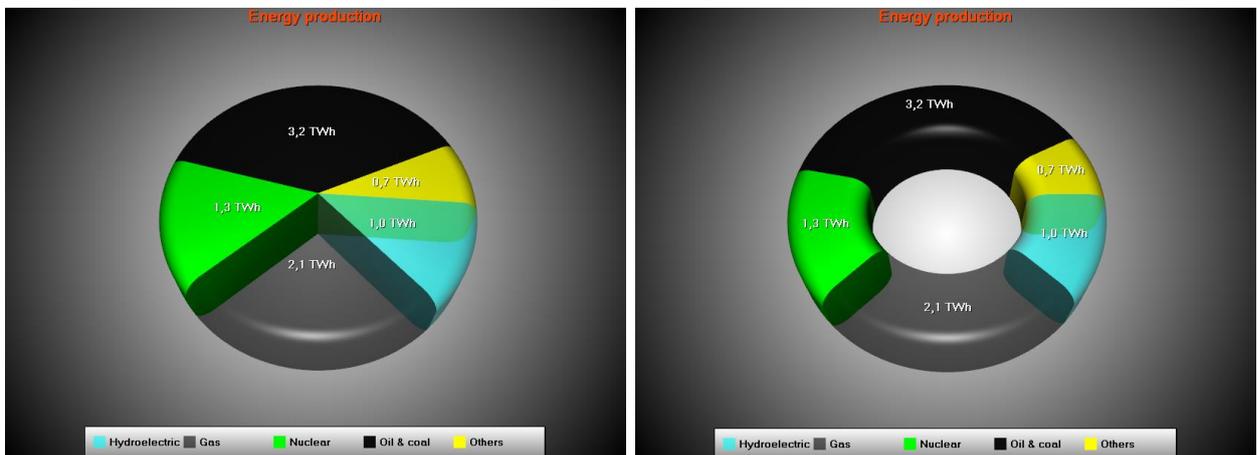


Figure 8-2. Example of a 3D Pie chart and a Donut chart.

8.1 Properties

Select the chart type, *Pie* or *Donut*, by using *Style* property. Control the zooming, panning and rotation with *ZoomPanOptions* property tree, similarly to View3D (see chapter 6.18).

Camera property controls the viewpoint (see chapter 6.4). Predefined lighting setup can be selected with *LightingScheme* property. Use *Material* property and its sub-properties to adjust general 3D surface appearance and shininess.

Use *DonutInnerPercents* to set the donut inner radius, *Rounding* to adjust edge rounding radius, *StartAngle* to rotate the pie, and *Thickness* to adjust pie thickness. *ExplodePercents* adjusts how far away the exploded pie slice is, when slice's *Explode* is set *true*.

TitlesStyle sets the pie slice text of one of the following: *Titles*, *Values* or *Percents*. Edit *TitlesNumberFormat* for example to "0.0 TWh" to include units in the end.

Annotations can be used as in View3D, but without axis value binding properties (see chapter 6.21).

8.2 Pie slices

Pie chart data is stored in *Values* collection. Each item in the list is of type *PieSlice*. Edit the data value in *Value* property. Set title string into *Title.Text* property. By defining *TitleAlignment* = *Outside*, the title is drawn outside the pie.

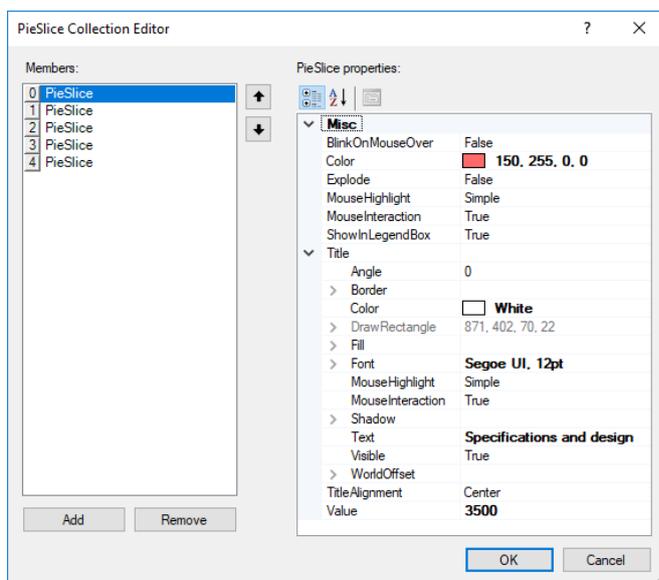


Figure 8-3. The Values list editor of a Pie chart.

8.3 Setting data by code

Data is stored in the **Values** list as **PieSlices**.

```
//Add pie slice data
//By using true as last parameter, the slice is automatically added to
chart.ViewPie3D.Values collection
PieSlice slice1 = new PieSlice("Hydroelectric",
Color.FromArgb(150, Color.Aqua), 1.0, chart.ViewPie3D, true);

PieSlice slice2 = new PieSlice("Gas",
Color.FromArgb(150, 0, 0, 0), 2.1, chart.ViewPie3D, true);

PieSlice slice3 = new PieSlice("Nuclear", Color.Lime, 1.3, chart.ViewPie3D,
true);

PieSlice slice4 = new PieSlice("Oil & coal", Color.FromArgb(240,0,0,0), 3.2,
chart.ViewPie3D, true);

PieSlice slice5 = new PieSlice("Others", Color.Yellow, 0.66,
chart.ViewPie3D, true);

slice3.Explode = true;
```

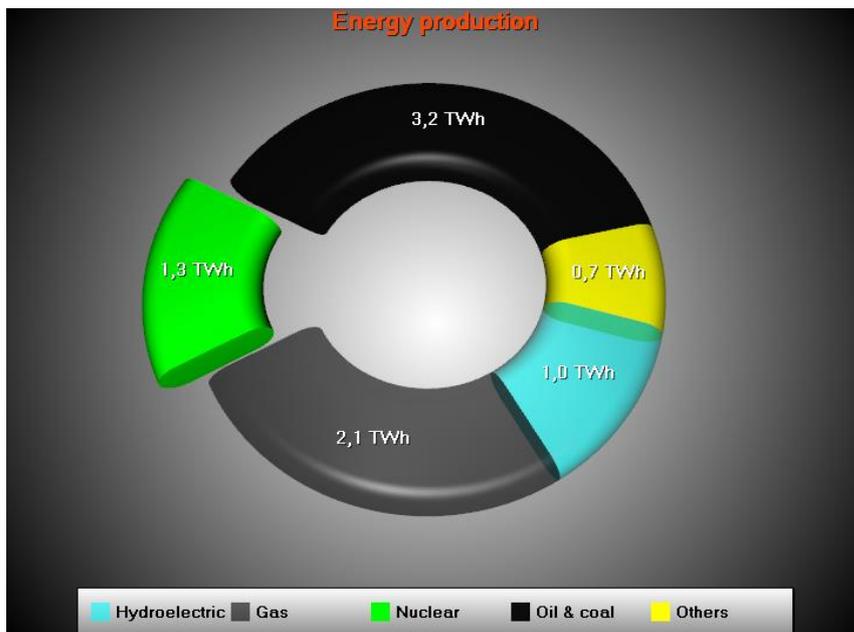


Figure 8-4. Data set into chart. Third slice is separated by using `slice3.Explode = true`.

8.4 Viewing pie chart in 2D

Set the camera as predefined camera from top.

```
chart.ViewPie3D.Camera.SetPredefinedCamera(PredefinedCamera.PieTop);
```

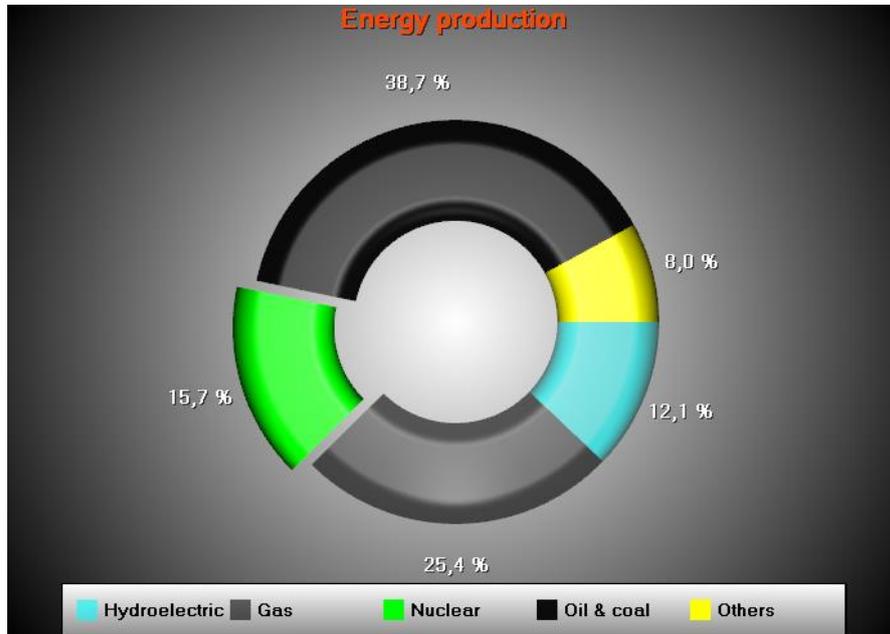


Figure 8-5. Pie chart shown as 2D, with a predefined camera from top.

9. ViewPolar

ViewPolar allows data visualization in a polar format. The data point position is determined by angular value and amplitude (compare angle as X and amplitude as Y in ViewXY). Polar view also has zooming and panning features.

ViewPolar	Polar chart view
Annotations	(Collection)
AreaSeries	(Collection)
AutoSizeMargins	False
Axes	(Collection)
AxisAutoPlacement	True
> Border	Border
> GraphBackground	
> LegendBox	LegendBoxPolar
> Margins	0, 0, 0, 0
Markers	(Collection)
PointLineSeries	(Collection)
Sectors	(Collection)
> ZoomCenter	0;0
> ZoomPanOptions	
ZoomScale	0.9223301

Figure 9-1. ViewPolar object tree.

9.1 Axes

Polar axes can be defined via **Axes** list property. Several axes can be used in same chart. Series can be assigned with any of these axes by setting **AssignPolarAxisIndex** property of a series. An axis represents both angular scale and amplitude scale. Otherwise, the polar axes are very similar to ViewXY axes (see chapter 5.2).

AmplitudeAxisAngle	0
AmplitudeAxisAngle Type	Relative
AmplitudeAxisLine Visible	True
AmplitudeLabelsAngle	0
AmplitudeLabels Visible	True
AmplitudeReversed	False
AngleOrigin	0
AngularAxisAutoDiv Spacing	True
AngularAxisCircle Visible	True
AngularAxisMajorDivCount	8
AngularLabels Visible	True
AngularReversed	False
AngularTicks Visible	True
AngularUnit Display	Degrees
AntiAliasing	True
AutoFormatLabels	True
AxisColor	 Sienna
Axis Thickness	4
> GridAngular	
GridVisibilityOrder	BehindSeries
InnerCircleRadiusPercentage	0
KeepDivCountOnRangeChange	True
> LabelsFont	Segoe UI, 9pt
LabelTicksGap	5
MajorDiv	6
MajorDivCount	5
> MajorDiv Tick Style	
> MajorGrid	
MarginInner	5
MarginOuter	5
MaxAmplitude	30
MinAmplitude	0
MinorDivCount	5
> MinorDiv Tick Style	
> MinorGrid	
MouseDownSnap ToDiv	False
MouseHighlight	Simple
MouseInteraction	True
MouseScaling	True
MouseScrolling	True
> ScaleNibs	
TickMark Location	Outside
> Title	RoundAxisTitle
> Units	RoundAxisTitle
UsePreviousAxisDiameter	False
Visible	True

Figure 9-2. AxisPolar property tree

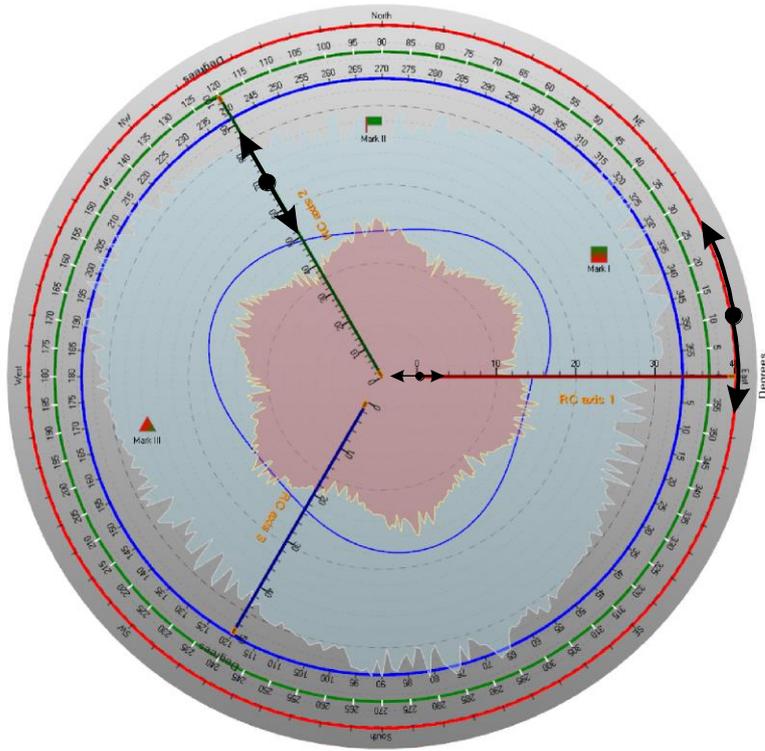


Figure 9-3. Three axes, the first one (red) in the outer circle, the second (green) in the middle, and the third (blue) closest to center. Axis AngleOrigin can be changed by dragging it over the axis circle. Amplitude range can be changed by dragging from the axis. Minimum or maximum of axis amplitude range can be changed by dragging from the small nib in the end of the amplitude scale.

9.1.1 Reversed axes

The axis can be reversed by amplitude, angle or both. To reverse the angle scale, set **AngularReversed = True**. To reverse the amplitude scale, set **AmplitudeReversed = True**.

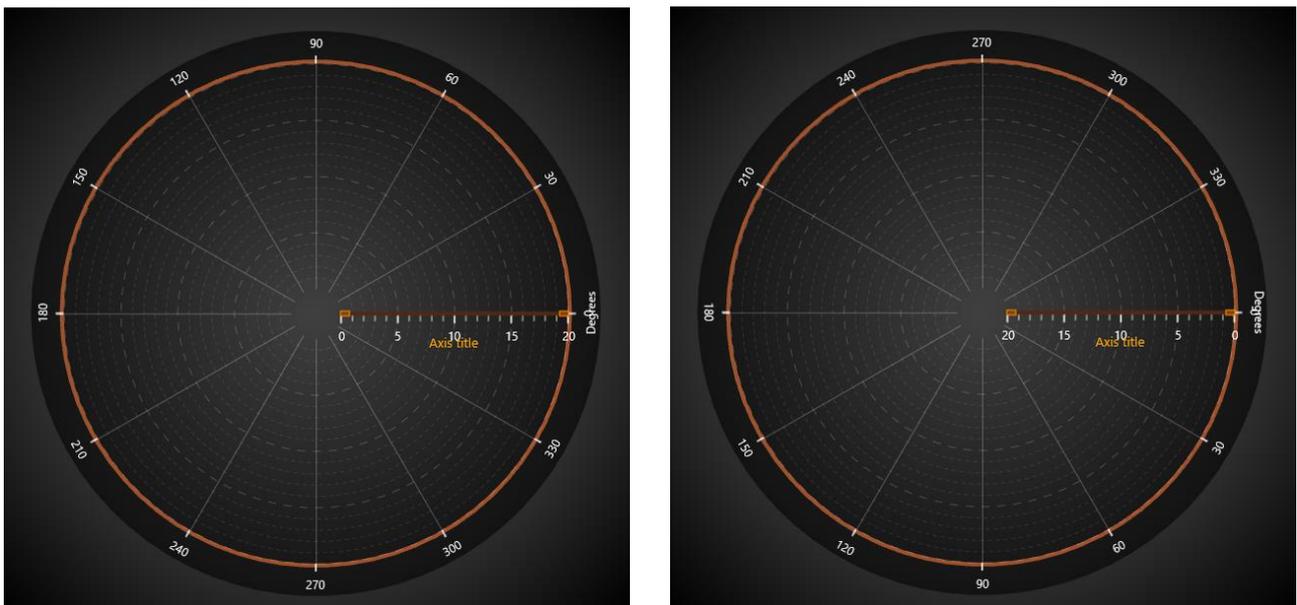


Figure 9-4. On the left, scales are not reversed. On the right, AngularReversed = True and AmplitudeReversed = True.

9.1.2 Setting rotation angles of the scales

Use **AngleOrigin** to set the rotation angle of angle scale.

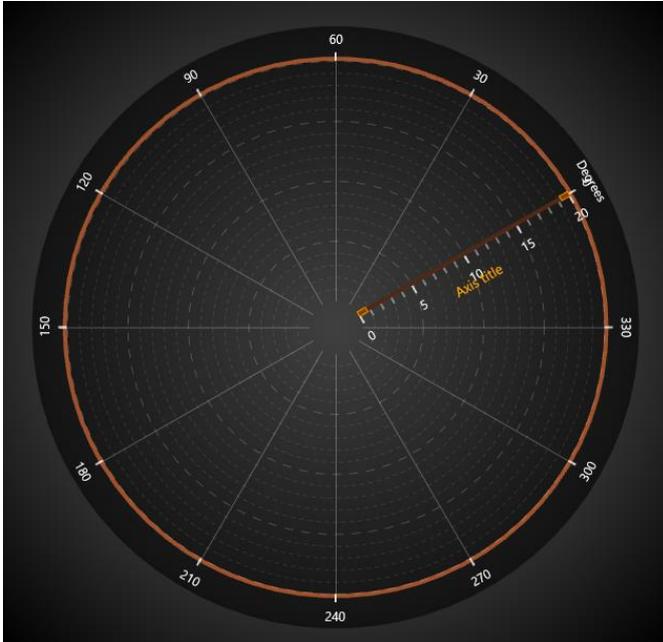


Figure 9-5. AngleOrigin = 30.

Use **AmplitudeAxisAngle** to rotate amplitude axis position. Amplitude scale angle can be set as absolute angle (**AmplitudeAxisAngleType = Absolute**), or relative (**AmplitudeAxisAngleType = Relative**) to angle scale's angle.

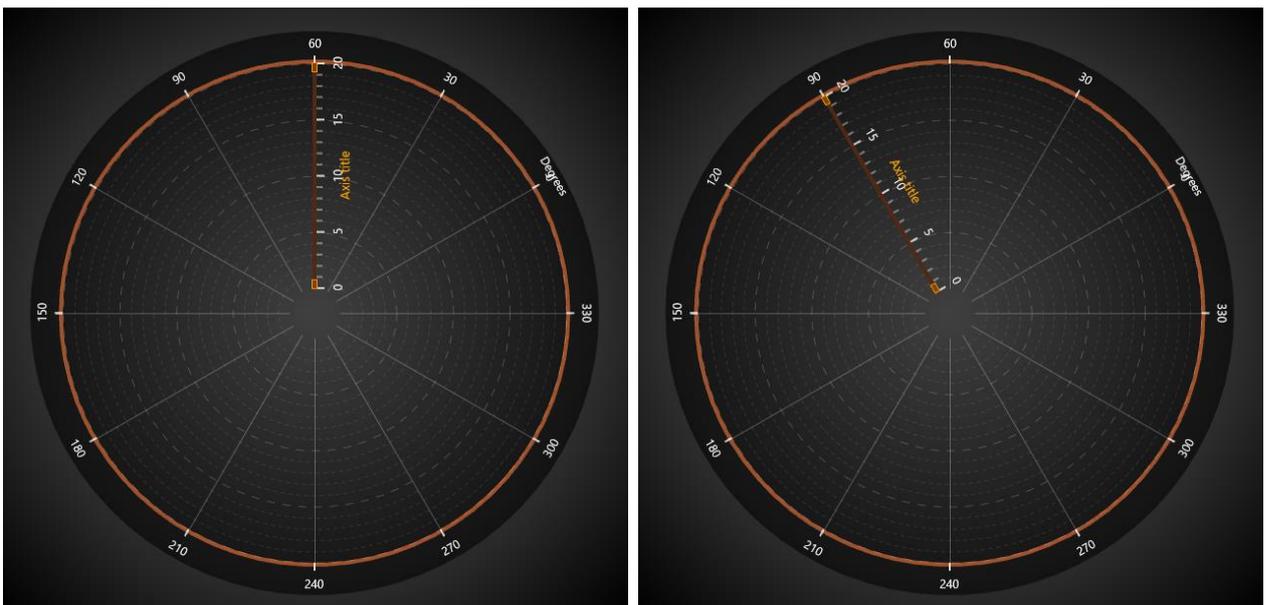


Figure 9-6. AngleOrigin = 30. AmplitudeAxisAngle = 90. On the left, AmplitudeAxisAngleType = Absolute. On the right, AmplitudeAxisAngleType = Relative. Overall the amplitude scale rotates 120 degrees in this case.

9.1.3 Setting divisions

Set the amplitude division count with **MajorDivCount**, and division magnitude with **MajorDiv** property. The amplitude scale will adjust accordingly (updating **MaxAmplitude**). Set amplitude minor division count with **MinorDivCount**.

By default, the chart tries to include almost as many angular divisions as it can fit. To control the angular divisions, set **AngularAxisAutoDivSpacing** to **False**. Then the chart tries **AngularAxisMajorDivCount** count of divisions. If chart space is too small to render all the divisions and labels, it will use a lower division count that it can fit.

9.2 Margins

When **AutoAdjustMargins** is **enabled**, the graph size is adjusted so that there's enough space for all the axes and chart title. When it is **disabled**, **ViewPolar.Margins** property applies allowing setting margins manually.

In the run time, the margins rectangle can be retrieved in pixels by calling **ViewPolar.GetMarginsRect** method, which applies to both automatic and manual margins. It is useful when needing to do screen-coordinate based computation or object placement.

ViewPolar.MarginsChanged event can be set to trigger when a margin rectangle has been changed because of for example resizing it.

The contents of the view are automatically clipped outside the margins. All contents are clipped other than the chart title, annotations and legend boxes as their position is defined in screen coordinates, allowing them to be freely positioned on the margins as well. A one-pixel wide border rectangle, **Border**, can be drawn to display where the margins are. By default, the border is not visible in ViewPolar. The color of the rectangle can be changed via **Border.Color**.

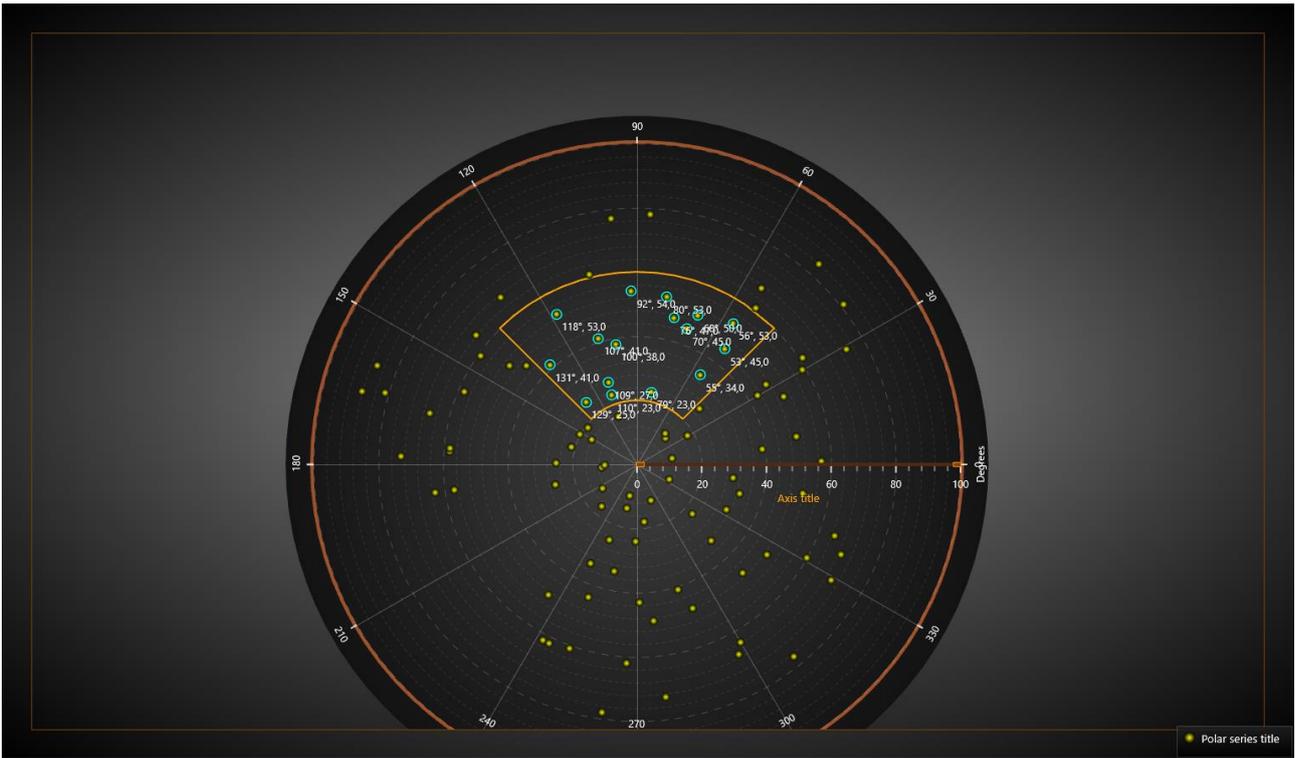


Figure 9-7. Contents of the polar chart are clipped outside the margins. Border is drawn to mark the margin area. Axis labels stay visible inside the borderline.

9.3 Legend boxes

Modify the legend box properties via ***ViewPolar.LegendBox***. Unlike *ViewXY*, *ViewPolar* can have only one legend box.

LegendBox	
AllowMouseResize	True
AutoSize	True
BorderColor	<input type="checkbox"/> 40, 255, 255, 255
BorderWidth	1
Categorization	None
CategoryColor	<input type="checkbox"/> White
> CategoryFont	Segoe UI, 10pt, style=Bold
CheckBoxColor	<input type="checkbox"/> 140, 255, 255, 255
CheckBoxSize	15
CheckMarkColor	<input type="checkbox"/> Khaki
> Fill	
Height	822
HighlightSeriesOnTitle	True
HighlightSeriesTitleColor	<input type="checkbox"/> Yellow
Layout	Vertical
MouseHighlight	Simple
MouseInteraction	True
MoveByMouse	True
MoveFromSeries Title	True
> Offset	
> PaletteScales	
Position	TopLeft
ScrollBarVisibility	Both
SeriesTitleColor	<input type="checkbox"/> White
> SeriesTitleFont	Segoe UI, 10pt
> Shadow	
ShowCheckboxes	True
ShowIcons	True
UseSeriesTitlesColors	False
Visible	True
Width	171

Figure 9-8. Legend box properties in *ViewPolar*.

9.3.1 Hiding palette scales

To hide the palette scale in a legend box, set ***PaletteScales.Visible = False***. To resize it, set ***ScaleSizeDim1*** and ***ScaleSizeDim2*** properties.

9.3.2 Legend box positioning in ViewPolar

ViewPolar's legend boxes can be placed automatically or manually. Automatic placement allows them to be aligned to the left/top/right/bottom side of the view, or the graph area. Control the position with **Position** property. Some positioning options take margin area into account while some do not.

Options ignoring margins (placing the legend box in the margin area):

TopCenter, TopLeft, TopRight, LeftCenter, RightCenter, BottomLeft, BottomCenter, BottomRight, Manual

Options placing the legend box inside the margin area:

GraphTopCenter, GraphTopLeft, GraphTopRight, GraphLeftCenter, GraphRightCenter, GraphBottomLeft, GraphBottomCenter, GraphBottomRight

Offset property shifts the position by given amount *from the position determined by Position* property.

```
// Setting legend box position, offset shifts from RightCenter position
chart.ViewPolar.LegendBox.Position = LegendBoxPosition.RightCenter;
chart.ViewPolar.LegendBox.Offset = new PointIntXY(-15, -70);
```

Manual positioning calculates the offset from the top-left corner of the legend box to the view's top-left corner. Note that this differs from **TopLeft** option, which is calculated from the top of the graph area.

9.4 PointLineSeries

ViewPolar's *PointLineSeries* can be used to draw a line, a group of points or a point-line. Lots of line and point styles are available in *LineStyle* and *PointStyle* properties.

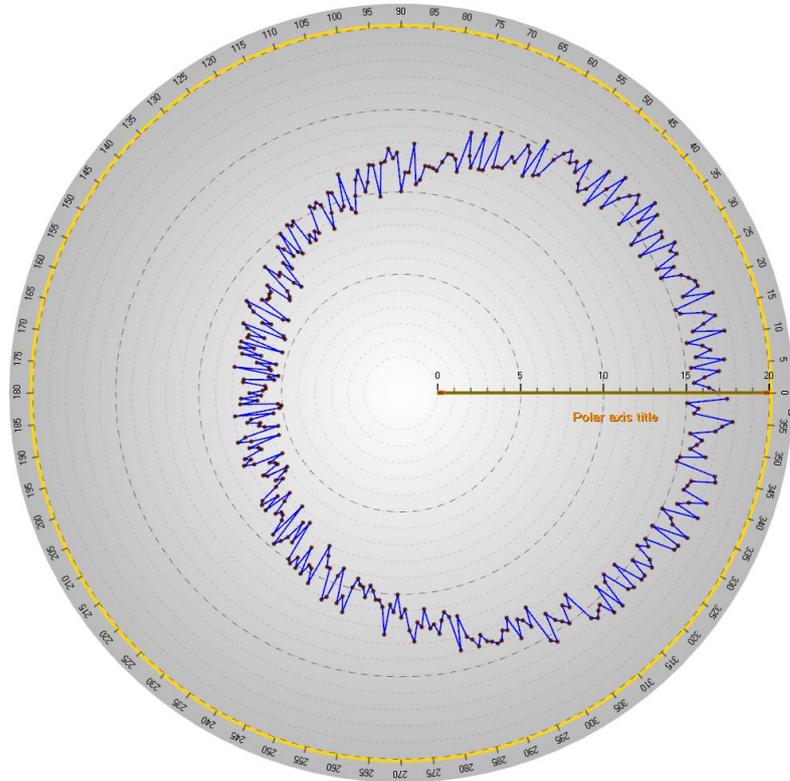


Figure 9-9. Some data presented with ViewPolar's *PointLineSeries*. Line and points are both visible.

9.4.1 Setting data

The code representing the data setting of the previous figure.

```
int iCount = 360;
PolarSeriesPoint[] points = new PolarSeriesPoint[iCount];
Random rnd = new Random();

for (int i = 0; i < iCount; i++)
{
    points[i].Amplitude = 10.0 + 3.0 * rnd.NextDouble() + 5.0 *
        Math.Cos(AxisPolar.DegreesAsRadians((double)i * 1.0));
    points[i].Angle = (double)i;
}
chart.ViewPolar.PointLineSeries[0].Points = points;
```

9.4.2 Palette coloring

Line coloring supports palette. **ColorStyle** property can be used to select how the palette coloring is applied.

- **LineStyle**: No palette fill. The color set in **LineStyle.Color** property applies
- **PalettedByAngle**: Data point **Angle** field determines the color
- **PalettedByAmplitude**: Data point **Amplitude** field determines the color
- **PalettedByValue**: Data point **Value** field determines the color

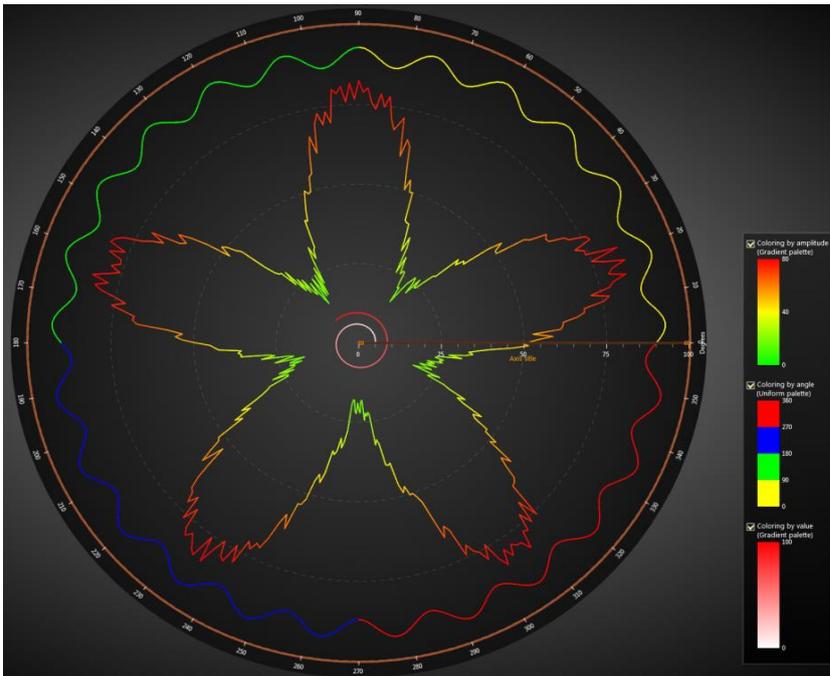


Figure 9-10. Palette coloring applied.

Use **ValueRangePalette** property to define the colors and value steps, it works similarly to **ViewXYs'** and **View3D's** series.

9.4.3 Custom shaping and coloring with CustomLinePointColoringAndShaping event

Custom coloring and coordinate adjustment can be made with **CustomLinePointColoringAndShaping** event, which is called just before entering the rendering stage of the chart. It works in similar way than the **CustomLinePointColoringAndShaping** event in **ViewXY's FreeformPointLineSeries** (see chapter 5.9.2).

9.5 AreaSeries

Area series allow data visualization in filled area style. The line style in the edge can be edited with **LineStyle** property. Fill can be changed with **FillColor** property.

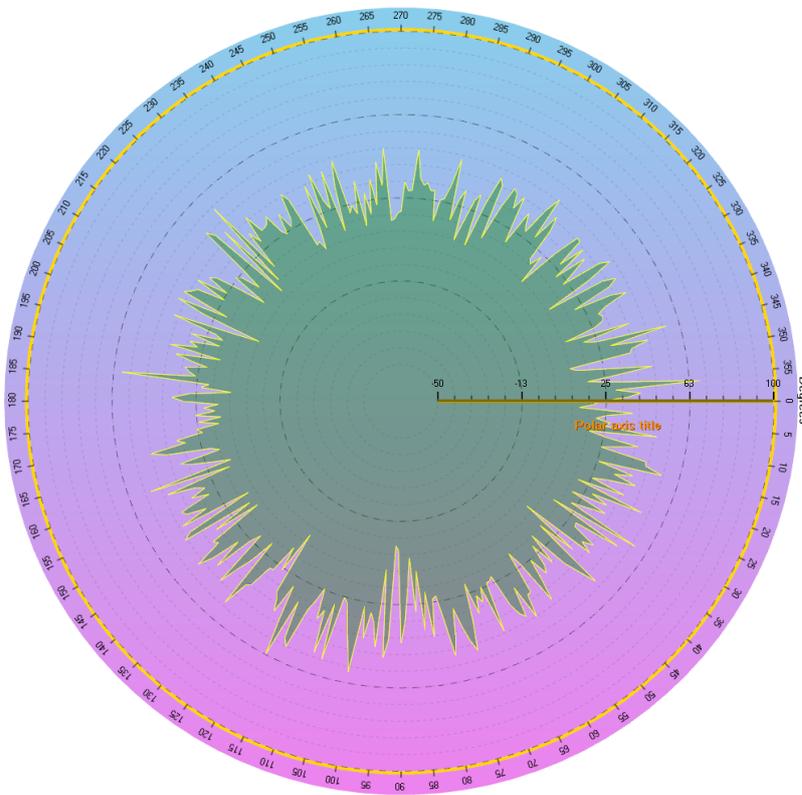


Figure 9-11. Some data presented with ViewPolar's AreaSeries.

9.5.1 Setting data

This code represents the data setting of previous figure.

```
int iCount = 360;
PolarSeriesPoint[] points = new PolarSeriesPoint[iCount];
Random rnd = new Random();

for (int i = 0; i < iCount; i++)
{
    points [i].Amplitude = 30f + rnd.NextDouble() * 5f *
        Math.Sin((double)i / 50f);
    points [i].Angle = (double)i;
}
chart.ViewPolar.AreaSeries[0].Points = points;
```

9.6 Sectors

Sectors can be defined to indicate some angular or amplitude range. Define amplitude range with **MinAmplitude** and **MaxAmplitude** properties. Define angular range with **BeginAngle** and **EndAngle**. Move a sector by dragging it with mouse.

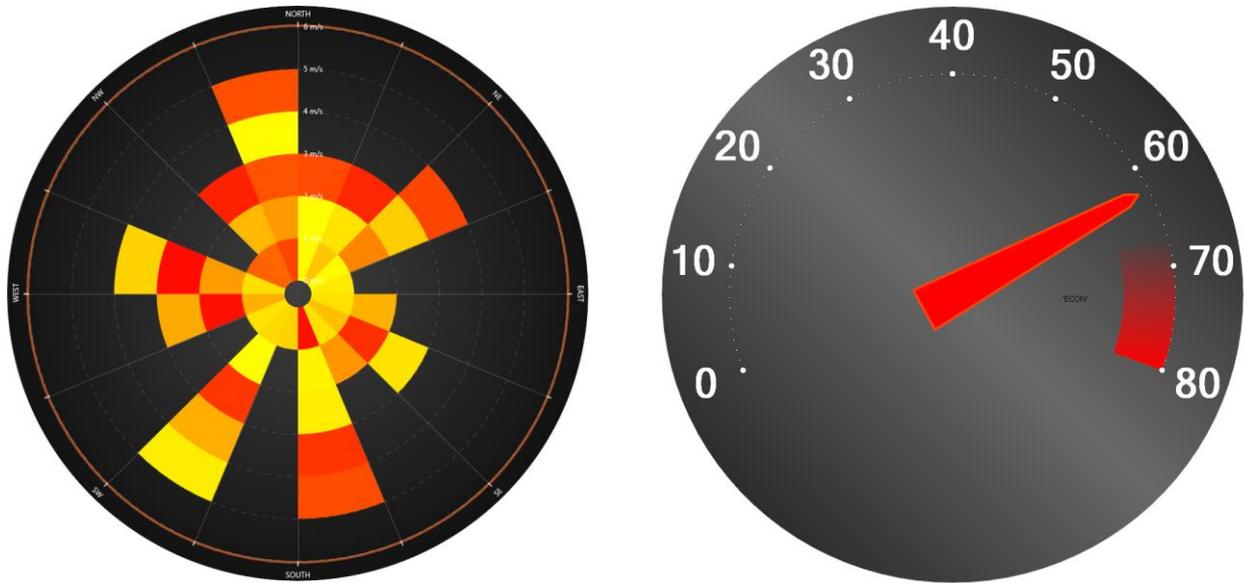


Figure 9-12. Two examples utilizing sectors. The first figure represents a PointLineSeries and a sector. In the second figure, a dial is made with AreaSeries with a sector representing RPM meter red zone.

9.7 Annotations

Annotations are similar to ViewXY's **Annotations** (chapter 5.20) with the exception of **Target** and **Location** being defined in Polar axis values. Sizing by axis values is not suitable and therefore **Sizing** property has only values **Automatic** and **ScreenCoordinates**.

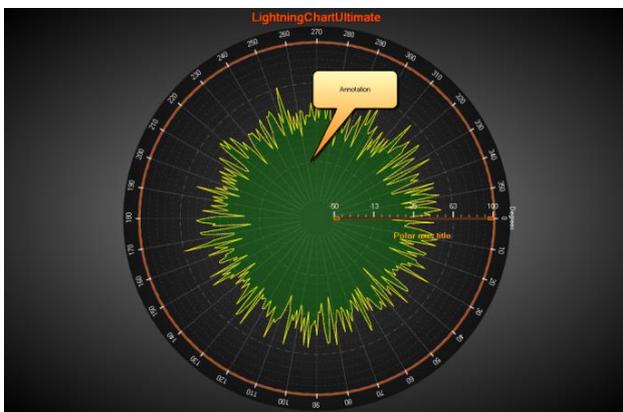


Figure 9-13. An annotation in Polar view.

9.8 Markers

Markers can be used to mark a specific data value at certain position. Assign the marker with a preferred axis by setting its **AssignPolarAxisIndex**. Define **Amplitude** and **AngleValue** properties to put it into place. Edit **Symbol** to have the preferred appearance and define the marker text with **Label** property.

Markers can be moved by dragging them with mouse. Set **SnapToClosestPoint** to **Selected** or **All** to enable nearest data point snapping when dragging it. **Selected** tracks only the series this marker is set to snap to with **SetSnapSeries()** method. **All** tracks all series.

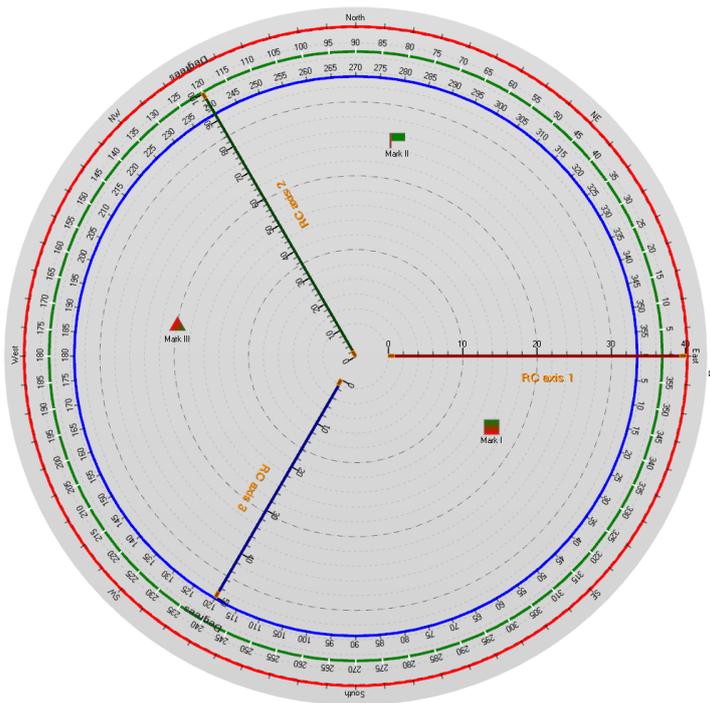


Figure 9-14. A couple of markers in a polar chart.

9.9 Zooming and panning

Zooming can be applied by code, by setting **ZoomCenter** and **ZoomScale** properties. **ZoomCenter** is defined as relative X-Y ranges.

X = -1: polar view's left edge at center of chart area

X = 0: polar view's center at center of chart area

X = 1: polar view's right edge at center of chart area

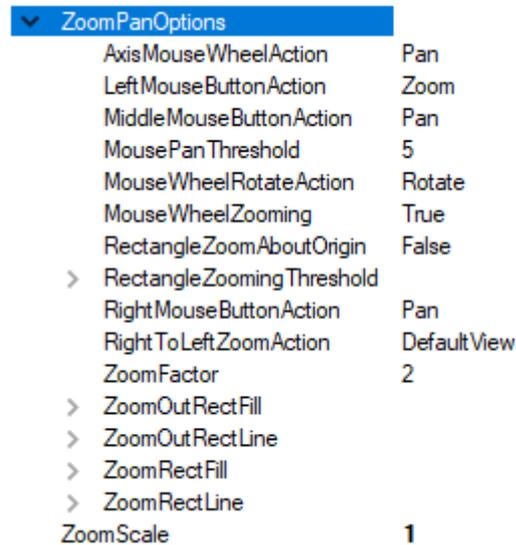
Y = -1: polar view's bottom edge at center of chart area

Y = 0: polar view's center at center of chart area

Y = 1: polar view's top edge at center of chart area

ZoomScale is the magnifying factor. For example, value 2 makes the chart appear twice as large in both X and Y direction compared to 1.

Mouse-zooming features can be configured in **ZoomPanOptions** property tree.



ZoomPanOptions	
AxisMouseWheelAction	Pan
LeftMouseButtonAction	Zoom
MiddleMouseButtonAction	Pan
MousePanThreshold	5
MouseWheelRotateAction	Rotate
MouseWheelZooming	True
RectangleZoomAboutOrigin	False
> RectangleZoomingThreshold	
RightMouseButtonAction	Pan
RightToLeftZoomAction	DefaultView
ZoomFactor	2
> ZoomOutRectFill	
> ZoomOutRectLine	
> ZoomRectFill	
> ZoomRectLine	
ZoomScale	1

Figure 9-15. ViewPolar's ZoomPanOptions.

9.9.1 Zooming operations and methods

Various zooming operations under **ZoomPanOptions** can be set as mouse actions. **DefaultSettings** returns the initial zoom and centering settings. **ZoomToData** (called **FitView** before v8.4) moves the view point to show all data inside the margins. **ZoomToLabelsArea** shows the whole data frame including labels inside the margins.

ViewPolar has the **ZoomPadding** property, which works similarly to View3D (chapter 6.18.3).

The zooming operations can also be accessed in code as methods by using **ZoomToFit(ZoomAreaRound.AreaName)**. For instance, calling **ZoomToFit(ZoomAreaRound.LabelsArea)** method gets the same result as performing **ZoomToLabelsArea** operation via mouse action.

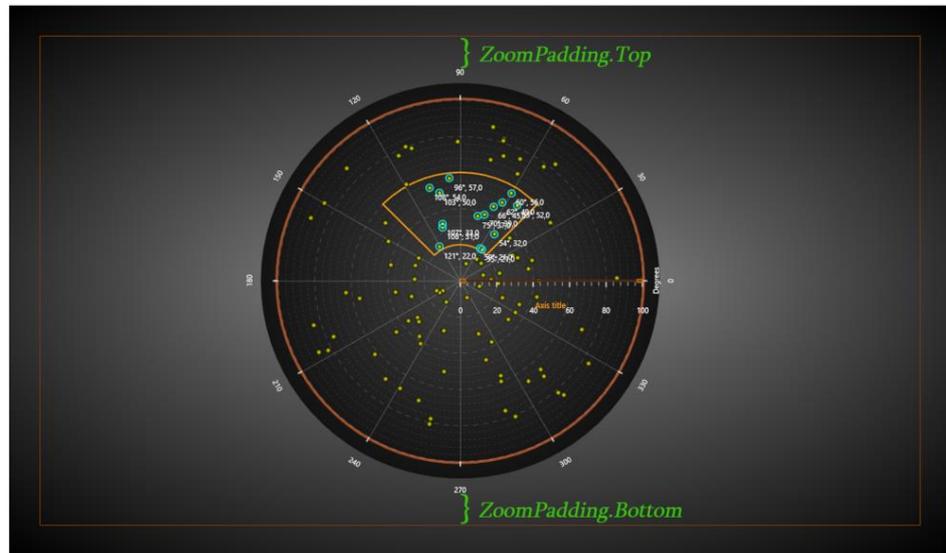
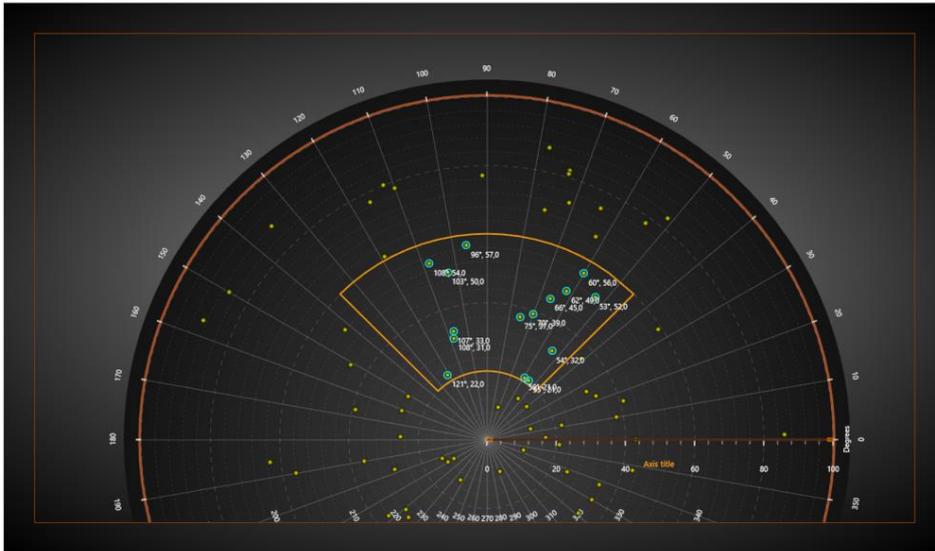


Figure 9-16. Polar chart before and after an zooming operation, `ZoomPadding = 50`. On top, the chart has been manually zoomed but no zoom operation has been called; `ZoomPadding` has no effect. Below, `ZoomToLabelsArea` was used, which takes into account the labels when zooming.

10. ViewSmith

Smith charts are generally used in electronics in impedance measurements and impedance matching applications.

Smith chart plots the data in real and imaginary values ($R + jX$).

Terms
Impedance = $Z = R + jX$
R = Resistance, Real part
X = Reactance, Imaginary part
X > 0: Capacitive
X < 0: Inductive

Data position is determined on 2D-plot by angular on circular Real and Imaginary log-log scales.

ViewSmith	Smith chart view
Annotations	(Collection)
AutoSizeMargins	False
> Axis	AxisSmithBase: Axis title
> Border	Border
> GraphBackground	
> LegendBox	LegendBoxSmith
> Margins	0, 0, 0, 0
Markers	(Collection)
PointLineSeries	(Collection)
> ZoomCenter	50;6,12303176911189E-15
> ZoomPanOptions	
ZoomScale	1

Figure 10-1. ViewSmith property tree.

10.1 Axis

The Smith chart has only one real axis, which can be configured via extended property tree **Axis**, see figure 10-2.

▼ Axis	
AngularAxisAutoDivSpacing	True
AngularAxisCircleVisible	True
AngularAxisMajorDivCount	8
AngularLabelsVisible	True
> AngularTickStyle	
AngularUnitDisplay	Degrees
AntiAliasing	True
AutoFormatLabels	True
AxisColor	 Sienna
AxisThickness	2
ClipGridInsideGraph	True
> GridAngular	
GridDivCount	5
GridDivSpacing	80
> GridImg	
> GridReal	
GridType	Distance
GridVisibilityOrder	BehindSeries
> LabelsFont	Segoe UI, 9pt
LabelTicksGap	5
MarginOuter	5
MouseHighlight	Simple
MouseInteraction	True
MouseScaling	True
RealAxisLineVisible	True
ReferenceValue	50
> ScaleNibs	
ShowAbsoluteValues	True
TickMarkLocation	Outside
> Title	
> Units	
Visible	True

Figure 10-2. Smith axis property tree.

Most of the properties are identical to ViewPolar's axes and ViewXY's axes to customize and make the chart more attractive. There are also advanced properties specific to ViewSmith adjustment, e.g. **GridDivCount**, **GridImg** and **GridReal**, **RealAxisLineVisible**, **ShowAbsoluteValues**, **ClipGridInsideGraph**.

GridDivCount defines the amount of circular grid lines on Real Axes and logarithmic grid lines on Imaginary scale.

GridImg and **GridReal** properties are responsible for customizing the grid lines either on **Real** or **Imaginary** scales. In addition, **Visible** property can be used to hide the grid, thus a user may hide one of them and continue to work with another.

RealAxisLineVisible property hides the axis line.

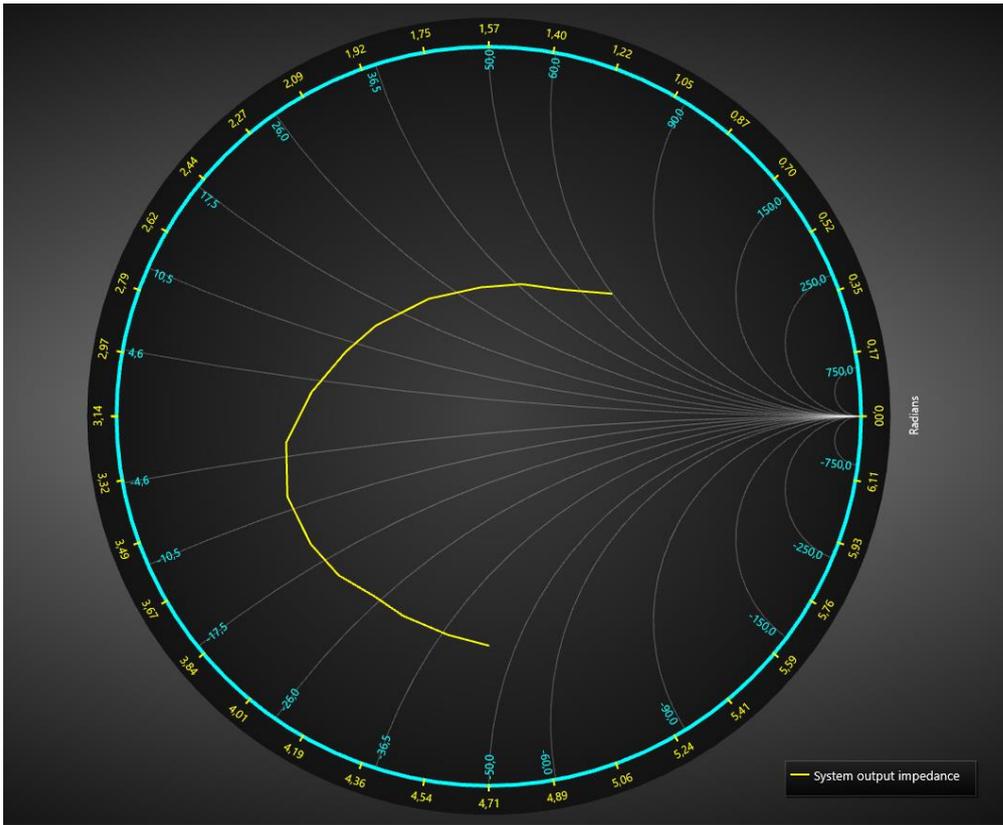


Figure 10-3. Real grid lines are hidden, Imaginary lines are visible.

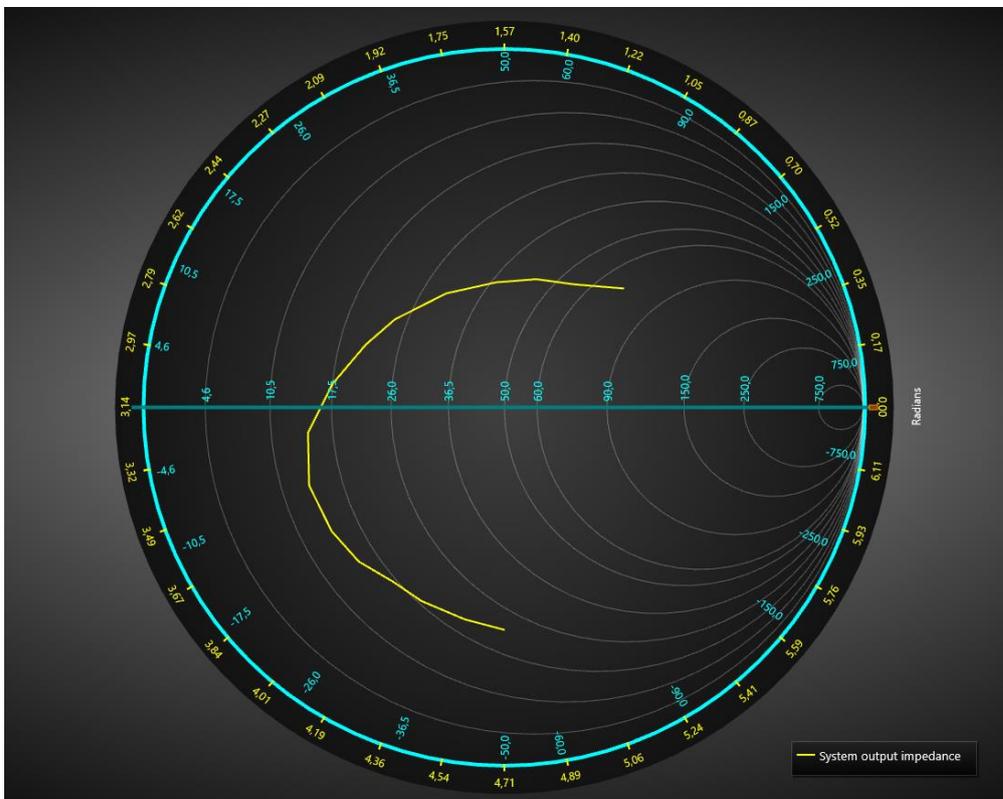


Figure 10-4. Imaginary grid lines are hidden, real lines are visible.

ShowAbsoluteValues property defines which values will be on scales (absolute or normalised).

ClipGridInsideGraph makes the gridlines visible outside the chart circle.

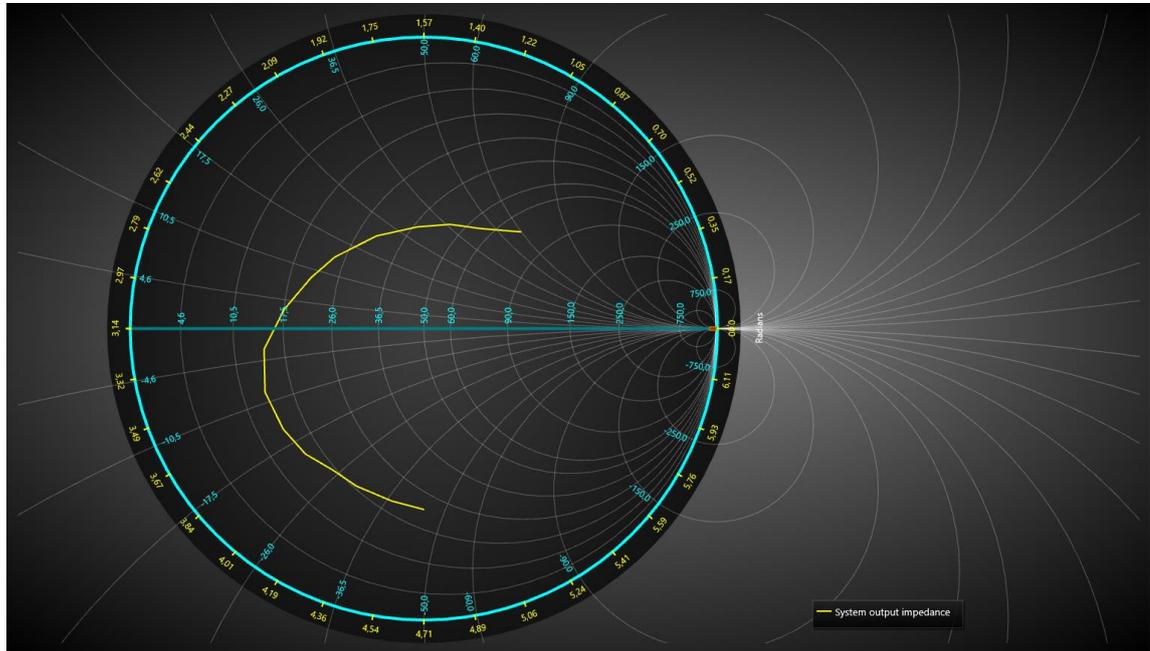


Figure 10-5. `ClipGridInsideGraph = False`.

The fully customized Smith chart can be seen below.

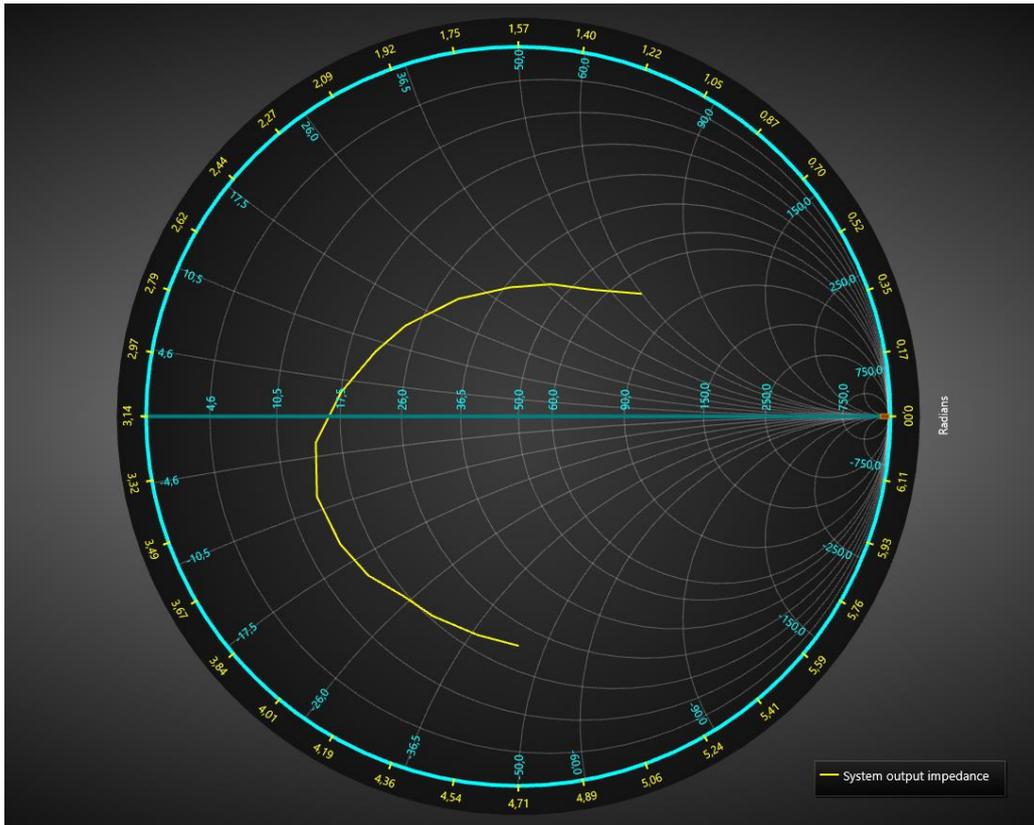


Figure 10-6. Customized Smith chart.

10.2 Margins

When **AutoAdjustMargins** is **enabled**, the graph size is adjusted so that there's enough space for all the axes and chart title. When it is **disabled**, **ViewSmith.Margins** property applies allowing setting margins manually.

In the run time, the margins rectangle can be retrieved in pixels by calling **ViewSmith.GetMarginsRect** method, which applies to both automatic and manual margins. It is useful when needing to do screen-coordinate based computation or object placement.

ViewSmith.MarginsChanged event can be set to trigger when a margin rectangle has been changed because of for example resizing it.

The contents of the view are automatically clipped outside the margins. All contents are clipped other than the chart title, annotations and legend boxes as their position is defined in screen coordinates, allowing them to be freely positioned on the margins as well. A one-pixel wide border rectangle, **Border**, can be drawn to display where the margins are. By default, the border is not visible in ViewSmith. The color of the rectangle can be changed via **Border.Color**.

10.3 Legend boxes

Legend boxes in ViewSmith work exactly like in ViewPolar (see chapter 9.3). Modify the legend box properties via **ViewSmith.LegendBox**.

10.4 PointLineSeries

ViewSmith's **PointLineSeries** can be used to draw a line, a group of points or a point-line as in ViewPolar. Lots of line and point styles are available in **LineStyle** and **PointStyle** properties.

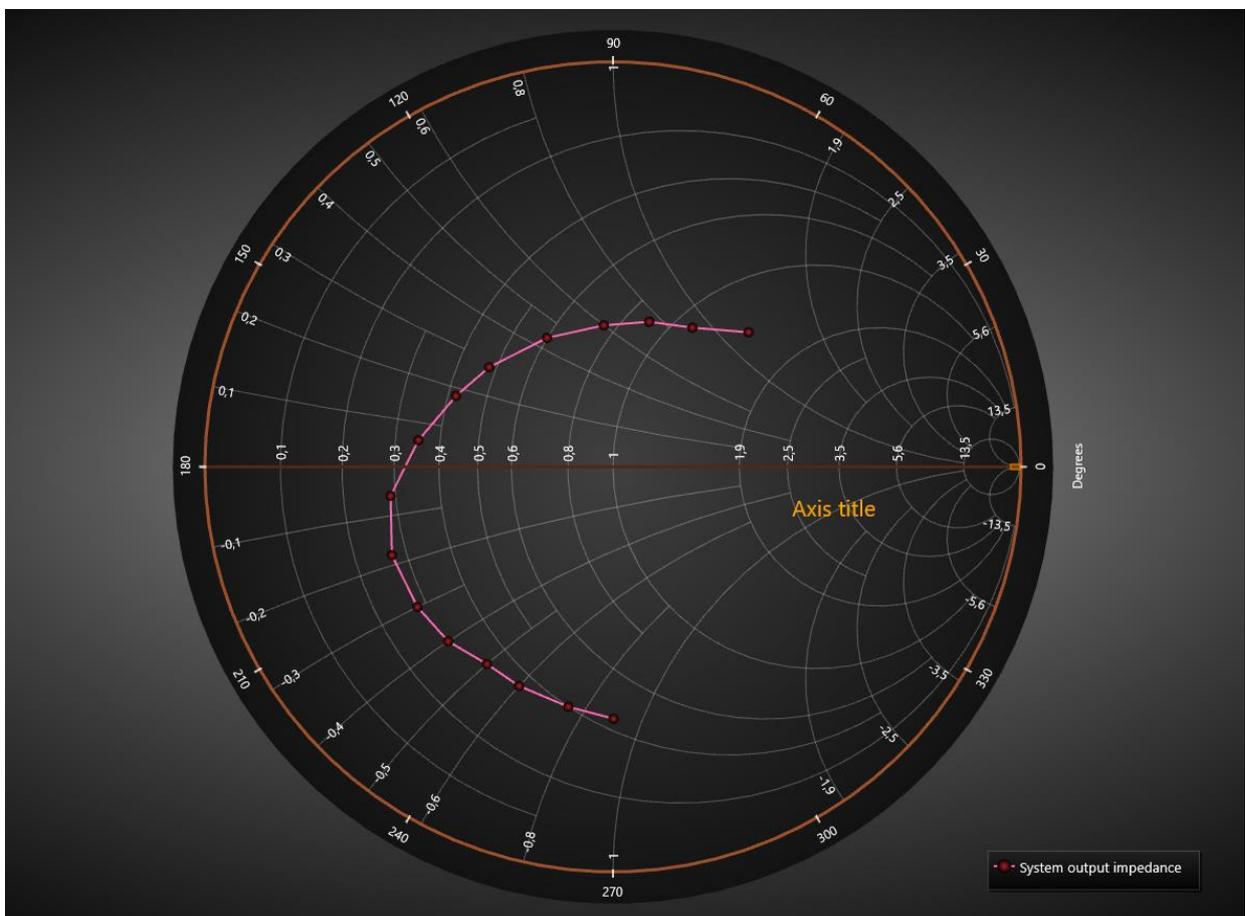


Figure 10-7. Smith data series.

10.5 Setting data

The code below, will add one set of data points to the collection of the Smith chart.

```
SmithSeriesPoint[] m_aPoints;
PointLineSeriesSmith Series = new PointLineSeriesSmith(m_chart.ViewSmith, axis);
//Create data for series
m_iCount = 5000;
m_aPoints = new SmithSeriesPoint[m_iCount];
for (int i = 0; i < m_iCount; i++)
{
    // Sine from left to right
    m_aPoints[i].RealValue = i * (MaxReal / m_iCount);
    m_aPoints[i].ImgValue = Math.Sin(0.01 * i)/Math.PI * MaxReal;
}
Series.Points = m_aPoints;
//Add series to chart
m_chart.ViewSmith.PointLineSeries.Add(Series);
```

10.6 Annotations

Annotations are identical to ViewXY's **Annotations** (chapter 5.20) except for **Target** and **Location** being defined in smith axis values (real and imaginary). **Sizing** property has only the values **Automatic** and **ScreenCoordinates**.

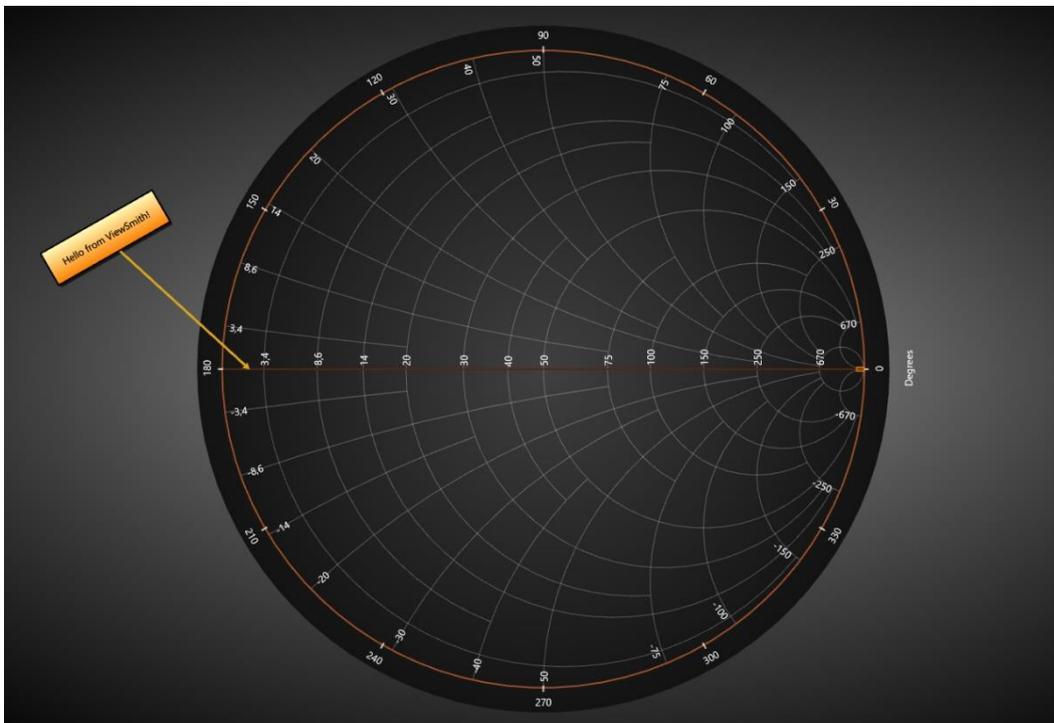


Figure 10-8. An annotation in ViewSmith.

10.7 Markers

Markers can be used to mark a specific data value at a certain position. Markers can be moved by dragging them with mouse. This property has identical definition with ViewPolar's markers (see chapter 9.8).

Define *ImgValue* and *RealValue* properties to position it. Edit *Symbol* to have the preferred appearance and define text with *Label* property.

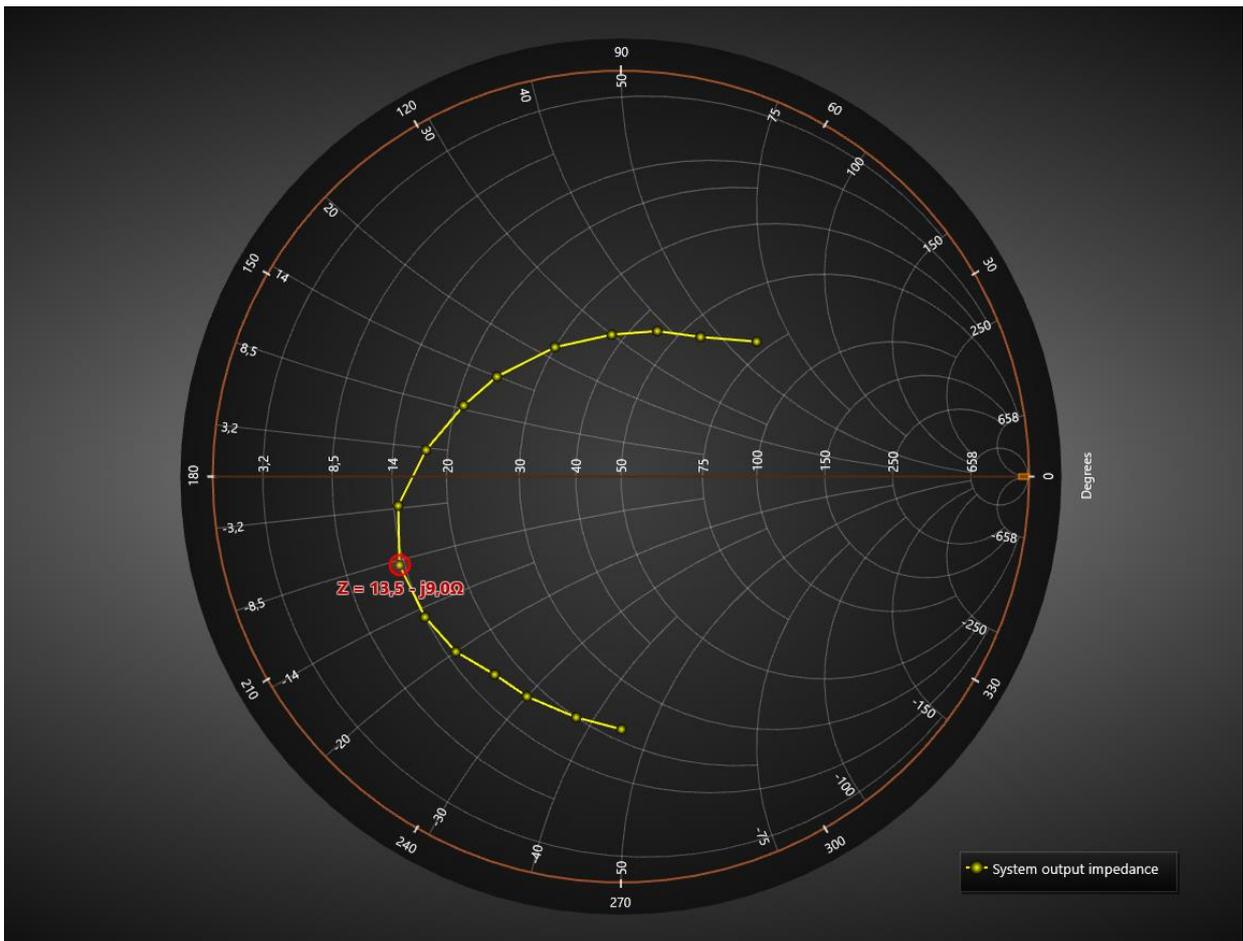


Figure 10-9. A marker tracking a series in Smith view.

10.8 Zooming and panning

Zooming and panning options and methods in ViewSmith work exactly the same as in ViewPolar (chapter 9.9).

11. Setting color theme

The overall color theme of a chart can be set with **ColorTheme** property. Setting the theme will override majority of the object colors in the created chart. It is advised to first set the **ColorTheme** and then modify the object colors.

```
chart.ColorTheme = ColorTheme.SkyBlue; // Changing the color theme
```

Note! By setting the color theme, every manually assigned color will be lost in the Visual Studio property grid without a warning.



Figure 11-1. Different color themes in use. On the left, default Dark theme with some custom colors. On the right, LightBlue theme set.

12. Scrollbars

One or more scrollbars can be added in *HorizontalScrollBars* or *VerticalScrollbars* collection property. The appearance is fully customizable, allowing defining even oval shaped buttons and scroll box. For example, a bitmap can be used as a button icon. Scrollbars can be used with all views, but the most apparent usage is in ViewXY.

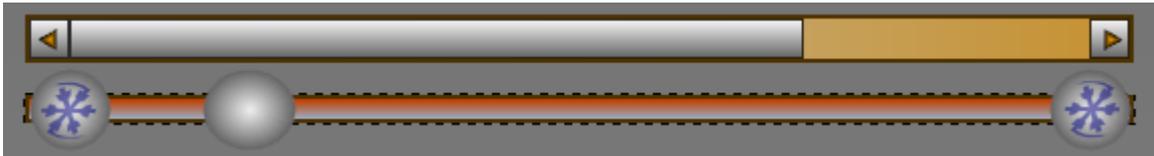


Figure 12-1. Two different looking scrollbars

HorizontalScrollBar can be aligned to fit the width of the graph by setting *Alignment* property to *BelowGraph*, *AboveGraph* or *GraphCenter*. Respectively, *VerticalScrollBar* can be aligned to fit the height of the graph by setting *Alignment* property to *LeftToGraph*, *GraphCenter* or *RightToGraph*. By setting *Alignment* to *None*, the scroll bar can be freely positioned with *Offset* property. Adjust its size with *Size* property.

The scrollbars have a 64-bit unsigned integer value instead of the usual 32-bit signed integer value range. *Value* is the current position, *Minimum* is the minimum range value and *Maximum* is the maximum range value. This allows direct support for long measurements with high sampling frequency. When *SampleDataSeries* is used in the measurement, set the sample index directly as scrollbar value. *Minimum* value represents the first sample index, and *Maximum* represents the last sample index.

SmallChange property is the amount of increment or decrement, when a scroll button is clicked. If *KeyControlEnabled* is active, you can use also arrow keys to change the *Value* by *SmallChange* amount. *LargeChange* represents a page change, which occurs when the scrollbar is clicked outside the scroll box or scroll buttons. Use *PageUp* and *PageDown* keys to change the *Value* respectively. *MouseWheelChange* sets the change value when mouse wheel is scrolled over the scroll bar.

Scroll event handler can be used in code to react to scrollbar value changes. Alternatively, *ValueChanged* event handler can be used. However, *Scroll* event handler provides more information of how the scroll has been done.

13. Export and printing

13.1.1 Bitmap image export

The chart can be exported as .PNG, .BMP and .JPG file with **SaveToFile()** method. **SaveToFile(...)** method allows exporting image files with resolution decrement and smoothing/anti-alias options. To export to a stream, use **SaveToStream()** method.

13.1.2 Vector image export

ViewXY, ViewPolar and ViewSmith can be also exported as .WMF, .EMF and .SVG formats. View3D and ViewPie3D don't currently support it. Use **SaveToFile** or **SaveToStream** method with selected vector file format.

Note! The vector output is simplified and all details, such as complex point styles, may be presented as a plain color and simple shape. The vector output may also contain some bitmap elements.

13.1.3 Copy to clipboard

The chart can be copied to clipboard by calling **CopyToClipboard(...)**. ViewXY, ViewPolar and ViewSmith can copied with **CopyToClipboardAsEmf()** method in vector format.

13.1.4 Capturing to byte array

The chart has **CaptureToByteArray** method, to get as a fast raw image data copy to external components or further processing of data.

Usage

```
int width;
int height;
byte[] aData = _chart.CaptureToByteArray(out width, out height);

Bitmap bitmap = new Bitmap(width, height,
    System.Drawing.Imaging.PixelFormat.Format32bppArgb);

System.Drawing.Imaging.BitmapData bitmapData = bitmap.LockBits(new
    System.Drawing.Rectangle(0, 0, width, height),
    System.Drawing.Imaging.ImageLockMode.ReadOnly,
    System.Drawing.Imaging.PixelFormat.Format32bppArgb);

IntPtr ipDst = bitmapData.Scan0;
int iRowByteCount = width * 4;
int iSrcIndex = 0;
for (int iY = 0; iY < height; iY++)
{
    Marshal.Copy(aData, iSrcIndex, ipDst, iRowByteCount);
    ipDst = new IntPtr(ipDst.ToInt64() + bitmapData.Stride);
    iSrcIndex += iRowByteCount;
}

bitmap.UnlockBits(bitmapData);
```

13.1.5 Setting output stream for continuous frame writing

Use **chart.OutputStream** property to set a stream into which the chart will write it's rendered frames.

This property is intended as the fastest way to capture continuous frames from the chart, especially on **Headless mode** (see chapter 22).

The stream is a raw byte stream, with each pixel described with 4 bytes, one byte per channel. The order of the channels depends on the renderer and its settings.

Use **GetLastOutputStreamFormat** and **GetLastOutputStreamSize** methods to find out the format and output size of the last written image.

Produced image size should be the size of the chart in pixels.

Note! On the contrary to the other properties of the LightningChart, set stream is NOT disposed on chart's dispose.

Note! On the contrary to the other properties, setting this property will not cause new frame to be rendered.

13.1.6 Printing

Call **PrintPreview()** method to open a print preview dialog or **Print()** to directly print with default settings. Call **Print(...)** to print with manual settings. Printing ViewXY, ViewPolar and ViewSmith supports vector printing as well. Supply **Raster** or **Vector** format to parameter to **Print(...)** method.

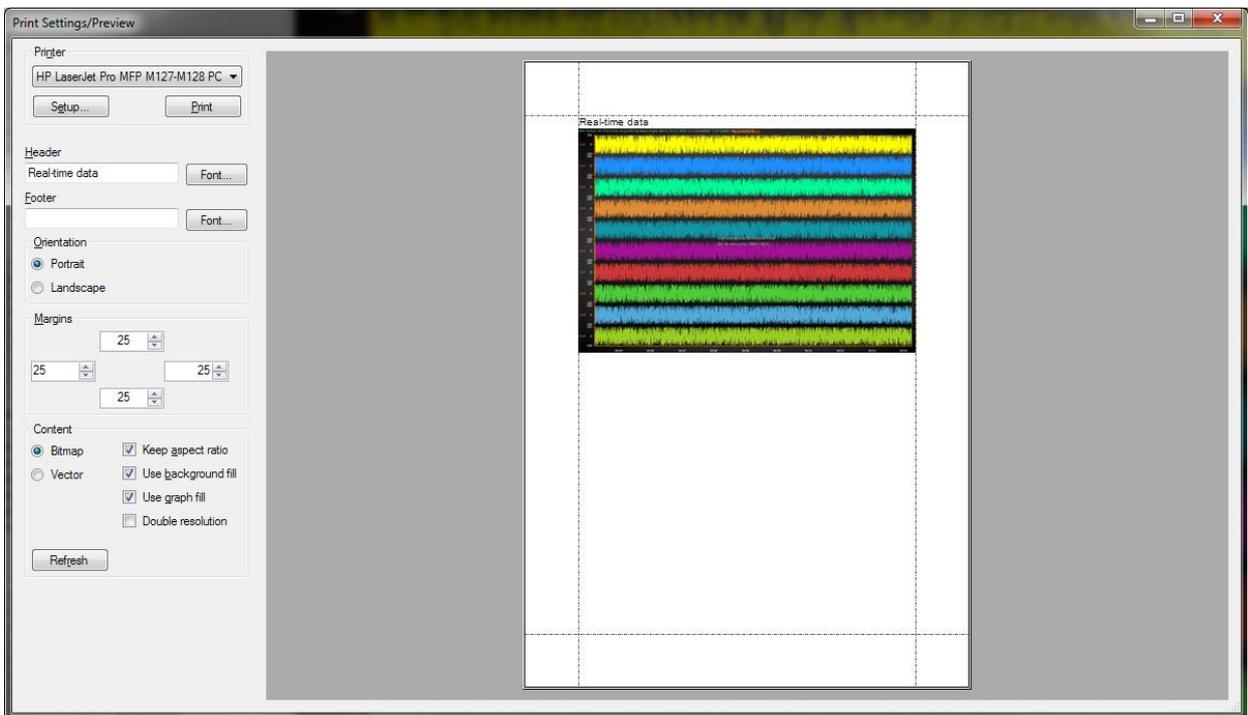


Figure 13-1. Print preview dialog.

14. LightningChart performance

14.1 Selecting the correct API edition

Select the chart edition as instructed in chapter 1.1. Do not use series data binding features unless necessary.

14.2 Set the rendering options correctly

LightningChart's DirectX9 rendering engine may be slightly faster than DirectX11 engine in specific applications, but generally, leaving DirectX11 as preferred renderer is a good choice. DirectX11 also gives better appearance.

Fonts quality setting is important as well.

See chapter 4.8.

14.3 Updating chart data or properties

Every property or series data value change will cause the LightningChart control to be redrawn. Every redraw will cause CPU and display adapter overhead. If more than one property is programmatically changed at the same time, the property changes should be made between ***BeginUpdate()*** and ***EndUpdate()*** method calls, as a batch. ***BeginUpdate()*** will stop drawing the control until ***EndUpdate()*** is called. There is an internal counter for pending ***BeginUpdate()*** calls, and when an equal amount of ***EndUpdate()*** calls have been reached, ***EndUpdate()*** redraws the control. The following example demonstrates how to update chart with minimal load to the computer.

```
chart.BeginUpdate(); //Disable redraws

//Add data to series
chart.ViewXY.SampleDataSeries[0].AddSamples(multiChannelSampleStream[0],
false);
chart.ViewXY.SampleDataSeries[1].AddSamples(multiChannelSampleStream[1],
false);
chart.ViewXY.SampleDataSeries[2].AddSamples(multiChannelSampleStream[2],
false);

//Update point counter bar
chart.ViewXY.BarSeries[0].SetValue(0,1,(double)totalPointsCollected,"",
false);
```

```

// Point counter label
chart.Title.Text = totalPointsCollected.ToString();

// Set monitoring scroll position to latest x
newestX = firstSampleTimeStamp + (double)(pointsLen - 1) / genSampFreq;
chart.ViewXY.XAxes[0].ScrollPosition = newestX;

chart.EndUpdate(); // Enable redraws and redraw

```

The internal counter allows using nesting updates as follows:

```

void MainMethod()
{
    chart.BeginUpdate();
    chart.Title.Text = "My title";
    chart.ViewXY.XAxes[0].AxisColor = Colors.Red;

    UpdateSeriesColors();

    chart.EndUpdate();
    // Repaints only once.
}

private void buttonCreate_Click(object sender, EventArgs e)
{
    UpdateSeriesColors(); // Repaints only once
}

void UpdateSeriesColors()
{
    chart.BeginUpdate();

    foreach(PointLineSeries series in chart.ViewXY.PointLineSeries)
    {
        series.LineStyle.Color = Color.Yellow;
    }

    chart.EndUpdate();
}

```

Updating series data depends on how the data is stored. For array series, **InvalidateData()** method has to be called after updating the array contents, otherwise the UI doesn't get notified about the changes.

ObservableCollection, useful for data binding in WPF semi-bindable and fully bindable, will update itself with every point or point's field because of automatic notifications. Thus, **InvalidateData()** is not needed. However, ObservableCollections cause slightly reduced performance compared to arrays or lists, which is noticeable especially when having a large amount of data points.

14.4 Line series tips

- When using a line series, use **SampleDataSeries**, if it is suitable for the application. It is the fastest one to draw and doesn't need as much memory as other line series types.
- Set **PointsVisible** property false, if the points don't have to be visible.
- Set line width to 1 with **LineStyle.Width** property.
- Use solid line style by setting **LineStyle.Pattern** to **Solid**.
- Disable anti-aliasing by setting line series **LineStyle.AntiAliasing** to **None** and set chart's **AntiAliasLevel** to 0.
- Disable all mouse interactivity, by setting **MouseInteraction** of a series to false. Alternatively, disable whole chart's mouse interactivity by setting chart's **chart.Options.MouseInteraction** to false.

14.5 Intensity series tips

Applies to: **IntensityGridSeries**, **IntensityMeshSeries**

- Change the **Optimization** property of the series to **StaticData**, if the data won't be updated continuously. **DynamicData** is better choice if data is changed many times per second.
- Use **Optimization: DynamicValuesData** to update only the **Value** fields of **Data** array's **IntensityPoint** structures, and call **InvalidateValuesDataOnly** method to update the chart. This way, the update is much faster as the geometry of the series is not recalculated. This is only intended to be used in applications where the data X and Y values of the nodes stay in the same place, for example in thermal imaging solutions.

Applies to: **IntensityGridSeries**

- For high-resolution thermal imaging applications, enable **PixelRendering** for **IntensityGridSeries**.
- For rapidly updating data sets, use **SetValuesData** and **SetColorsData** methods instead of **Data** property to save memory and to improve performance.

14.6 3D Orthographic view tips

Use **View3D.Camera.Projection = Orthographic** instead of **OrthographicLegacy** for maximum performance. Difference can be seen especially when zooming the view while having lots of series and data in it (see chapter 6.4).

14.7 3D surface series tips

Applies to: *SurfaceGridSeries3D*, *SurfaceMeshSeries3D*, *WaterfallSeries3D*

- Disable lighting by setting *SuppressLighting* to false if light reflections and shading are not needed.
- If contour lines are used, use *FastColorZones* or *FastPalettedZones* instead of *ColorLines* or *PalettedColorLines*.

Applies to: *SurfaceGridSeries3D*

- With scrolling data (like 3D spectrum or spectrogram), use *InsertRowBackAndScroll* and *InsertColumnBackAndScroll* methods to update data and axis ranges.

14.8 Maps tips

Applies to: *ViewXY.Maps*

- Set *ViewXY.Maps.Optimization* to *CombinedLayers* in a typical situation where the X and Y axis ranges are kept the same, and other data is presented over the maps. This allows the map layers to be rendered into the same buffer image, resulting into more efficient rendering.
- Set *ViewXY.Maps.Optimization* to *None*, if the map titles should be displayed over *IntensityGrid* or *IntensityMesh* series.

14.9 Hardware

To get the absolute maximum performance for a LightningChart application, the computer hardware must be powerful. In many applications, display adapter power is more important than CPU power. Use as modern display adapter as possible. DirectX 9.0c level display adapters work. 'c' comes from DirectX Shader Model 3, which is required by some effects.

GetRenderDeviceInfo() method can be called to find out if some feature is not supported by the used display adapter. Especially, if the returned information states that *FastVertexFormat* is not supported, it is a bad thing for performance.

Note! LightningChart is a GPU hardware accelerated chart. Without a good GPU, the performance may be much lower than in optimal case. A good resource to compare the performance of different GPUs is **PassMark's Video Card Benchmarks** (http://www.videocardbenchmark.net/gpu_list.php). A video card having 10x better score than other, can be also 10 times faster in LightningChart use, but overall difference in refresh rate is rarely that great, since another computer hardware may become a bottleneck.

15. LightningChart notifications, error and exception handling

From version 8.4 onwards LightningChart will send messages from the chart to the user through **ChartMessage** event. The messages can contain notifications about for example chart performance, incorrect usage, warnings or errors. Define a handler for **chart.ChartMessage** event to listen to the messages. The event contains a **ChartMessageInfo** struct, which holds the message's information.

Prior to version 8.4 chart sent messages through **ChartError** event (now marked as obsolete), which contains less information than **ChartMessage**. User can listen to **ChartError** events instead of **ChartMessages** and get the same base information, but it is recommended to use **ChartMessage** instead.

ChartMessageInfo's MessageSeverity property tells how severe the message is. Messages can be filtered based on their severity. Possible severity levels for messages are:

- **Debug** - Debug information which usually is not interesting to the user and no action is required.
- **Information** – An incorrect usage of a chart, for example using an invalid property setting, has happened which should not impact chart performance. User action is typically not required.
- **Warning** – Some incorrect usage of chart, for instance using a disposed object, has happened which might cause some minor problems with the chart such as performance loss. User action might be required.
- **RecoverableError** – An error has occurred from which the chart should have recovered. User must listen to **ChartMessage** events or messages with this severity will be thrown as exceptions.
- **UnrecoverableError** – An error has occurred from which the chart couldn't recover. Might indicate an incoming exception. User must listen to **ChartMessage** events or messages with this severity will be thrown as exceptions.
- **Critical** – A critical error has occurred in the chart which will always be thrown as an exception.

MessageType property explains the basic type of the message while **Details** property has more specific information about it. All the possible message types can be found in **MessageType** enum located in LightningChart namespace.

Unwanted messages can be filtered out by changing the **chart.Options.ChartMessageMinimumLevel** property value. The property allows only messages of the set minimum level and higher to be sent through the event system. It is set to **MessageSeverity.Warning** by default.

Exceptions are thrown as **ChartException** objects, which contains **ExceptionInfo** struct with detailed information about the exception similar to **ChartMessage** events. In some cases, the chart may throw exceptions of other types, such as a rendering engine exception. If user wants the chart to raise an exception on all messages with a severity level of **MessageSeverity.Warning** or higher, **chart.Options.ThrowChartExceptions** property needs to be set **true** (is **false** by default).

It is recommended to always subscribe to **ChartMessage** event to be notified about errors in the chart and possible exceptions from unlistened messages. In case of having any problems with the chart and support for it is needed, please ensure there is a working message/exception handler in the application, log the **ChartMessages** and include them in your support request.

16. ChartManager component

ChartManager control can be used to coordinate interoperation of several LightningChart Ultimate controls. Add **ChartManager** control to your form. Then, assign the manager control to **ChartManager** property of all LightningChart Ultimate controls.

16.1 Chart interoperation, drag-drop

ChartManager enables series drag-drop from chart to another, in WinForms. For WPF it's not usable for technical reasons.

Series have **DisableDragToAnotherAxis** property which must be set to **False** to enable the dragging. It is **True** by default.

Axes have **AllowSeriesDragDrop** property which can be set to **False** to prevent dragging over specific axis. Default value is **True**.

Move mouse over the series to be dragged, press left mouse button down to start dragging.

Dragging over Y axis: Drag the series over Y axis of another chart and release the button. The other chart takes the ownership of the series and the series is assigned to the target Y axis. This also assigns **the first X axis** for the series.

Dragging over X axis: Drag the series over X axis of another chart and release the button. The other chart takes the ownership of the series and the series is assigned to the target X axis. This also assigns **the first Y axis** for the series.

16.2 Memory management enhancement

In some extreme real-time monitoring applications, the .NET garbage collector does not free unused memory well enough, if the application is run with high CPU load. Garbage collector frees all the memory at once, causing a visible 'freeze' or 'pause' when updating a chart. To make the chart updates smoother, enable ChartManager's **MemoryGarbageCollecting** property. This allows a separate thread to be used to free the memory more often regardless of the CPU load. Using **MemoryGarbageCollecting** is recommended to be used with multi-core processors, as the thread running will slightly load the CPU.

17. SignalGenerator component

SignalGenerator component can be used to generate real-time signal. The signal is produced as the sum of different waveforms. Several **SignalGenerator** components can be linked by master-slave relationship, to produce a synchronized, multi-channel output. **SignalGenerator** is very useful when developing signal monitoring or data acquisition software with LightningChart.

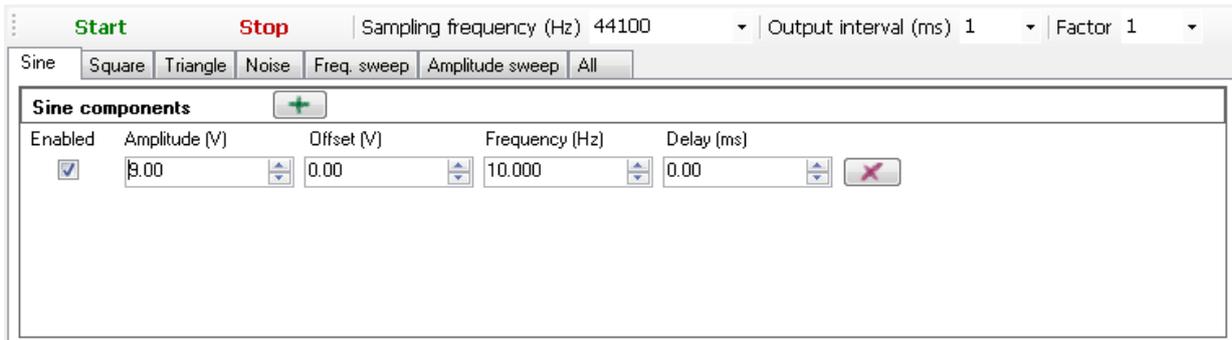


Figure 17-1. Signal generator component with Sine page selected.

The waveforms are divided into following categories: **Sine**, **Square**, **Triangle**, **Noise**, **Frequency sweep**, and **Amplitude sweep**. Respective tab pages for them can be seen in the component. In Sine page, sine waveforms can be added. In Square and Triangle pages, square and triangle waveforms can be added respectively. In Noise page, random noise waveforms can be generated. In Frequency and Amplitude sweep pages, frequency and amplitude sweeps can be added. In **All** page, all the waveforms can be set in a stacked view.

17.1 Sampling frequency, Output interval and Factor

SamplingFrequency tells how many signal points are generated per second. Higher sampling frequency produces more accurate signal but comes with a cost of increased dataflow and overhead. With high sampling frequency, signals containing high frequencies can be presented. Sampling frequency must be more than twice the maximum signal frequency to fulfill Nyquist sampling theorem.

OutputInterval sets the preferred interval of calculated output samples, in milliseconds. For example, if **OutputInterval** is set 100, a bundle of samples is received 10 times per second, after every 100 ms period. Using lower values will give smoother real-time monitoring output. Note that **OutputInterval** is not accurate and may vary with computer load. The output data stream automatically generates more samples if the period has been longer than expected. The data stream will get in shape also with high data rates and under heavy computer overhead.

Factor multiplies the output samples by selected value. For example, to generate mV signal instead of V signal, set **Factor** to 1E-3.

17.2 Sine waveforms

Sine waveform is constructed with **Amplitude**, **Offset**, **Frequency** and **DelayMs** parameters. **Amplitude** is the maximum voltage difference from zero level. Note that the total range is bipolar. Peak-to-peak value will be $2 * \text{Amplitude}$. **Offset** is DC level added to the signal. In other words, positive values shift the signal up and negative values shift it down in the value range. **Frequency** tells the signal cycle count in Hertz. One cycle per second is frequency of 1 Hertz. **DelayMs** delays in the signal in milliseconds.

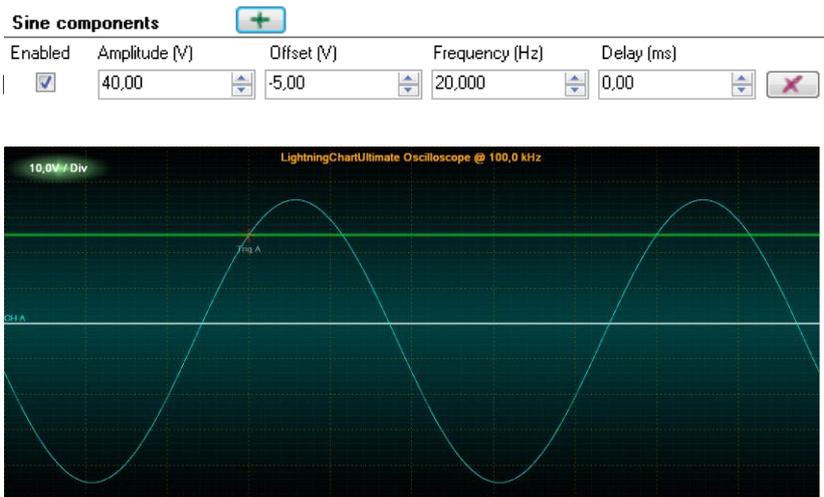


Figure 17-2. A simple sine waveform signal with settings above.

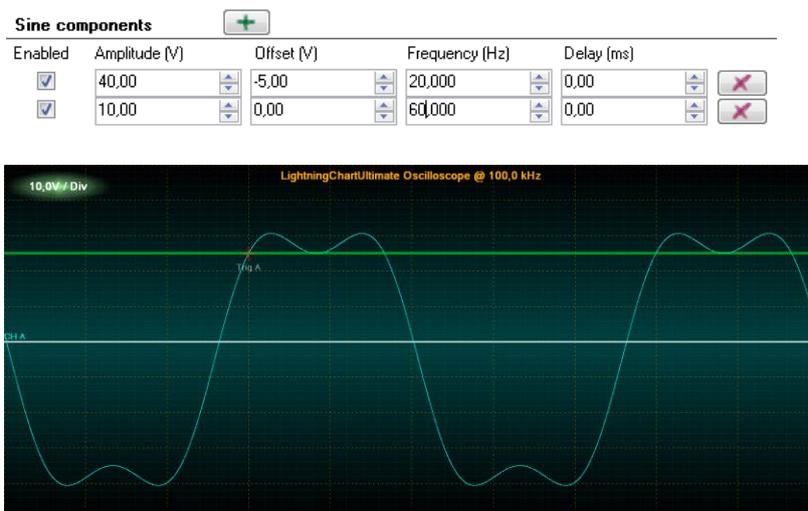


Figure 17-3. The signal of two sine waveforms with their settings above.

17.3 Square waveforms

Square waveform has one more parameter compared to sine waveforms, **Symmetry**. The range for **Symmetry** is 0...1. **Symmetry** tells how long the signal stays in high state, related to cycle period. With a value of 0.5 the low and high states of the signal are of equal lengths.

Square components

Enabled	Amplitude (V)	Offset (V)	Frequency (Hz)	Delay (ms)	Symmetry
<input checked="" type="checkbox"/>	35,00	0,00	30,000	0,00	0,80

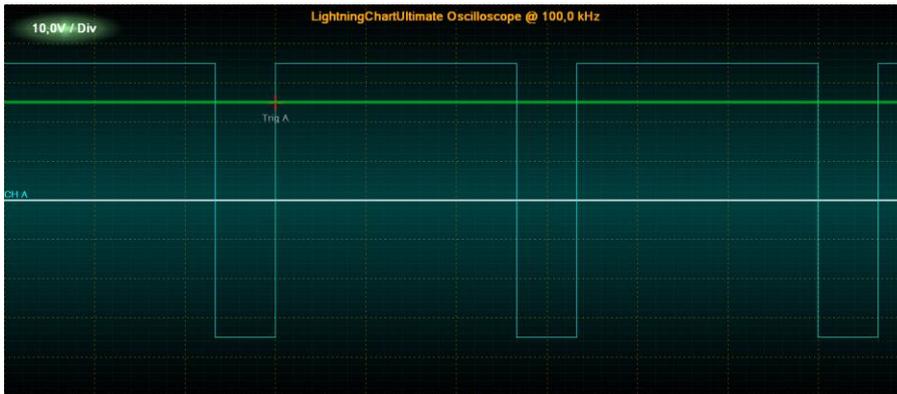


Figure 17-4. One square waveform signal, with symmetry of 0.8. Settings above.

17.4 Triangle waveforms

Triangle waveform also has **Symmetry** parameter. It controls the way the triangles lean. 0.5 is the value for symmetrical triangle. Values under 0.5 lean left and values over it lean right.

Triangle components

Enabled	Amplitude (V)	Offset (V)	Frequency (Hz)	Delay (ms)	Symmetry
<input checked="" type="checkbox"/>	40,00	0,00	25,000	0,00	0,70

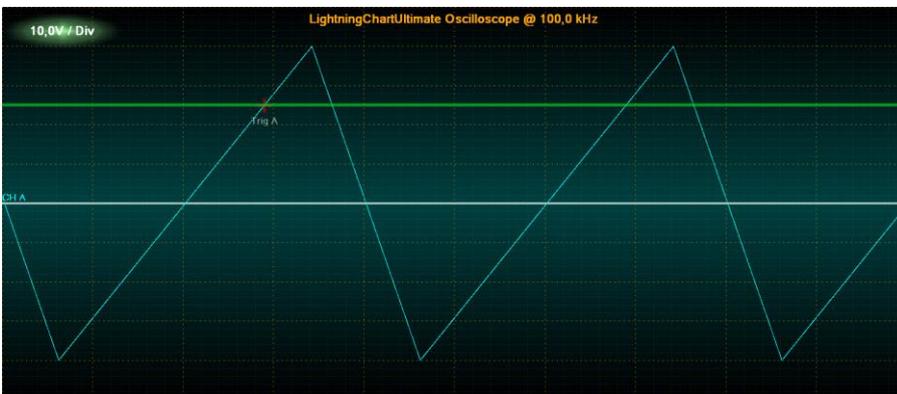


Figure 17-5. One triangle waveform signal, with symmetry of 0.7.

17.5 Noise waveforms

Noise waveform is a randomly generated signal. Points get randomized between **-Amplitude** and **+Amplitude**.

Noise components

Enabled	Amplitude (V)	Offset (V)
<input checked="" type="checkbox"/>	30,00	0,00

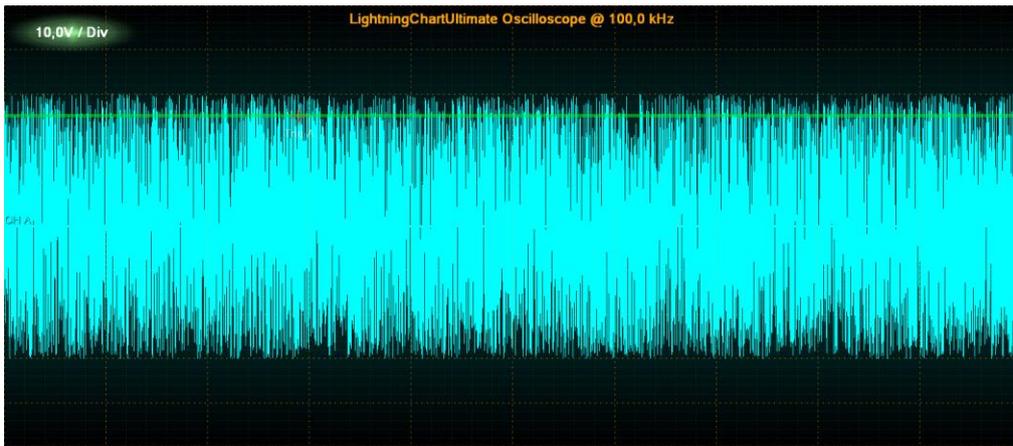


Figure 17-6. Noise waveform signal generating random data points between amplitudes -30 and +30.

17.6 Frequency sweeps

Frequency sine sweep runs from frequency 1 to frequency 2 in given time period, with constant amplitude. Use **FrequencyFrom** to set the start frequency, **FrequencyTo** to set the end frequency, **Amplitude** to set the constant amplitude, and **DurationMs** to set the duration in milliseconds.

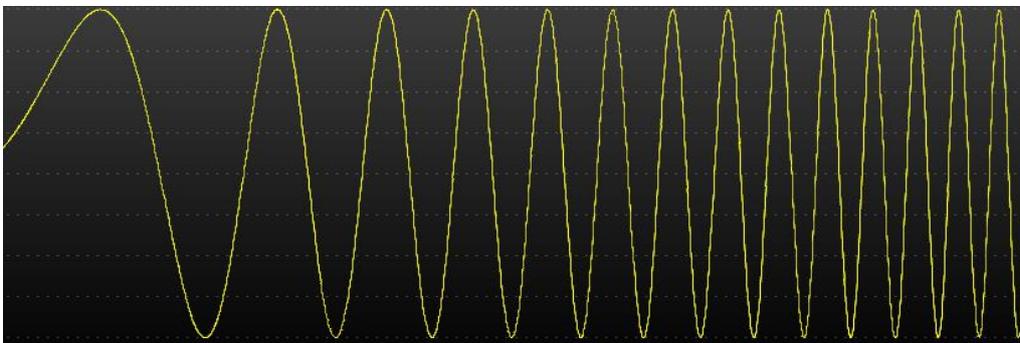


Figure 17-7. Frequency sweep.

17.7 Amplitude sweeps

Amplitude sine sweep runs from amplitude 1 to amplitude 2 in given time period, with constant frequency. Use **AmplitudeFrom** to set the start amplitude, **AmplitudeTo** to set the end amplitude, **Frequency** to set the constant frequency, and **DurationMs** to set the duration in milliseconds.

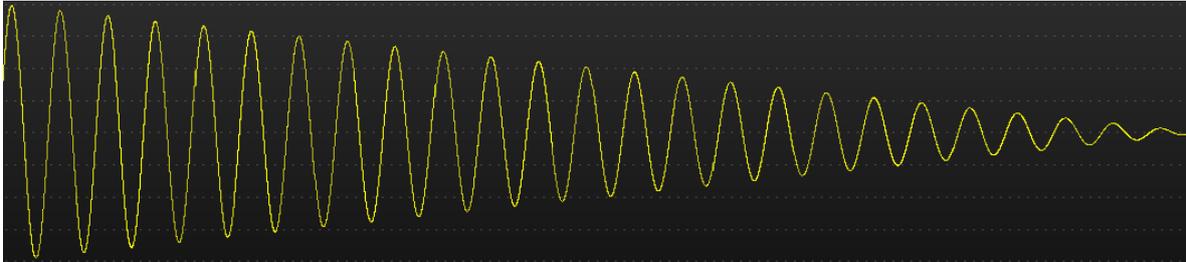


Figure 17-8. Amplitude sweep.

17.8 Starting and stopping

Start the generator by pressing Start button or calling **Start** method. Stop the generator by pressing Stop button or calling **StopRequest** method. **Stopped** event will fire when stopping is complete.

17.9 Multi-channel generator with master-slave configuration

Several **SignalGenerator** components can be connected to produce a synchronized, multi-channel output.

MasterGenerator controls the sampling frequency, start, stop and output of all generators. It produces the first channel in the output data stream.

Slave generators are connected to master generator by assigning their **MasterGenerator** property. Define the signal waveforms freely. Slave generators are started and stopped by the master generator. They get the output data stream channel index in the connection order. Slave generators must be connected before starting the master generator.

17.10 Output data stream

The output data stream consists of two-dimensional arrays, obtained via **DataGenerated** event handler. Generally, the event is raised after every **OutputInterval**.

The event handler obtains a reference to a samples array, and a time stamp for the first samples bundle received during this interval. The first dimension of samples array represents channels and the second the samples for each channel. All channels have equal sample count.

Raising **DataGenerated** event:

```
m_signalGenerator.DataGenerated += m_signalGenerator_DataGenerated;

private void m_signalGenerator_DataGenerated(DataGeneratedEventArgs args)
{
    // Event code
}
```

To investigate the channel count of the data stream, get the length of first dimension:

```
channelCount = args.Samples.Length;
```

To get the sample count of a channel:

```
sampleBundleCount = args.Samples[0].Length;
```

The following code will demonstrate how to forward the output data directly to **SampleDataSeries** list of LightningChart while updating real-time monitoring scroll position.

```
private void m_signalGenerator_DataGenerated(DataGeneratedEventArgs args)
{
    chart.BeginUpdate();
    int channelIndex = 0;
    int sampleBundleCount = args.Samples[0].Length;
    foreach (SampleDataSeries series in chart.ViewXY.SampleDataSeries)
    {
        series.AddSamples(args.Samples[channelIndex++], false);
    }

    //Set latest scroll position x
    newestX = args.FirstSampleTimeStamp + (double)(sampleBundleCount - 1) /
    generatorSamplingFrequency;
    chart.ViewXY.XAxes[0].ScrollPosition = newestX;
    chart.EndUpdate();
}
```

Note that with **args.Samples[0]** you can access the master generator's data. **args.Samples[1]** gives access to first slave generator data, **args.Samples[2]** to second slave etc.

18. SignalReader component

SignalReader component allows reading data from a signal source file and playing it back with selected rate. **SignalReader** output data stream format is similar to **SignalGenerator** (see chapter 17.10).

SignalReader component currently supports wav and sid formats.

18.1 Key properties

FileName defines the file to be opened, for example "c:\\wavedata\\audioclip1.wav"

Factor sets the output factor. Raw signal samples are multiplied by this value.

OutputInterval is similar to **SignalGenerator's** property (see chapter 17.1).

IsLooping allows file read to jump to the beginning of the file, when the end of file has been reached.

After the file has been opened, the following properties can be used to get information of the file:

ChannelCount: the channel count of the file.

SamplingFrequency: sampling frequency in Hz.

FileSize: File size in bytes.

Length: Sample count for each channel. It may not be exact for all signal file formats.

IsReaderEnabled: Status telling is the component started and reading data. If **Looping** is set to false and end of file is reached, **IsReaderEnabled** will change to false.

18.2 Opening file quickly for playback

Call **OpenFile(...)** method supplied with a file name. The file name must have an extension of supported formats. Then, call **Start()** method.

```
signalReader.OpenFile("c:\\wavedata\\audioclip1.wav");  
signalReader.Start();
```

A playback of a PCM-formatted WAV file is then started

The playback can be stopped by calling **StopRequest()** method.

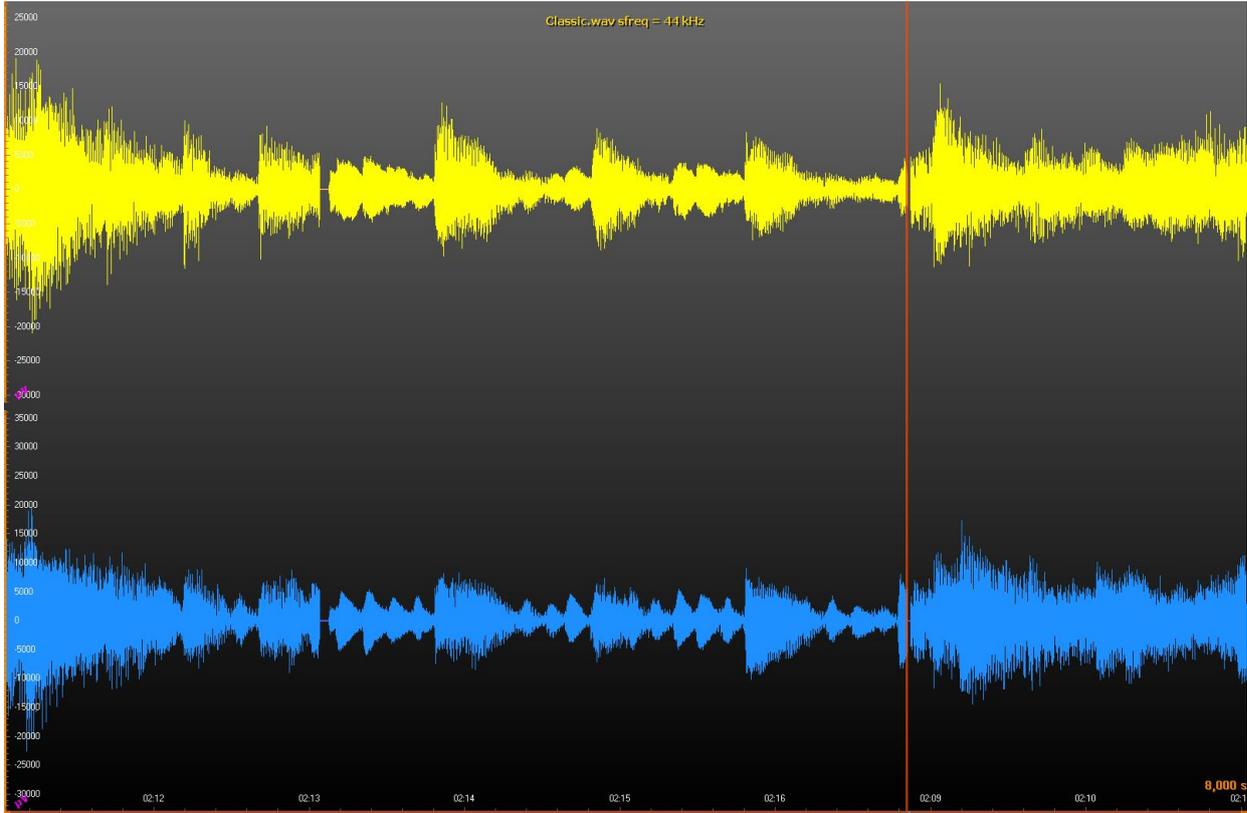


Figure 18-1. SignalReader reads a wav file and LightningChart SampleDataSeries draw the signal. A cursor line is used to mark the current reading position and the X-axis scroll position.

19. AudioInput component

AudioInput component allows user to capture signal from Windows' recording device to System.Double values. These values can then be rendered on LightningChart Ultimate, sent to an AudioOutput component, saved to a file etc...

19.1 Properties

BitsPerSample – Gets or sets how many bits are allocated per sample. Supported values are 8 and 16. If other value is used, 16 is used instead. It can be set when **IsInputEnabled** is **false**.

IsInputEnabled – Gets or sets the state of this instance (i.e. starts or stops it). Setting this property **true** is the same as calling **Start** method where **false** is the same as calling **Stop** method.

IsStereo – Gets or sets whether to use two channels (stereo) or just one (mono). Can be set when **IsInputEnabled** is **false**.

LicenseKey – Gets or sets the license key in normal or encrypted format.

RecordingDevice – Gets or sets the current recording device. Can be set when **IsInputEnabled** is **false**. By setting this property to **null**, Windows' default recording device is used.

SamplesPerSecond – Gets or sets the sampling frequency. Can be set when **IsInputEnabled** is **false**.

ThreadInvoking – Gets or sets whether this instance automatically synchronizes its events to the main UI thread, hence eliminating the need to call **Control.Invoke** method on caller's side.

Volume – Gets or sets the volume, ranging from 0 to 100. Can be set when **IsInputEnabled** is **false**.

19.2 Methods

GetRecordingDevices – Use this static method to get a list of available Windows recording devices.

RequestStop – Signals this **AudioInput** instance to stop. Stop does not occur immediately after exiting this method. By subscribing to **Stopped** event, caller is notified when everything has stopped.

Start – Starts reading audio from selected recording device. **Started** event is triggered when internal thread is about to start.

19.3 Events

DataGenerated – Occurs when a new set of audio data has been generated. Data and its first sample's time stamp can be read from a **DataGeneratedEventArgs** object that is provided as a parameter.

Started – Occurs when an audio input has been started. **StartedEventArgs** object that is provided as a parameter, contains three public fields: **BitsPerSample**, **ChannelCount** and **SamplesPerSecond**.

Stopped – Occurs when the audio input has been stopped.

19.4 Usage (WinForms)

This chapter describes the usage of WinForms version of **AudioInput** class. WPF version will be handled in chapter 19.5.

19.4.1 Creation

Create a new **AudioInput** instance either manually in the source code or by dragging and dropping it from Visual Studio's toolbox on to the form, user control etc.

If there is no need to show the GUI (i.e. if using an own GUI or controlling **AudioInput** object from the source code) then set **Visible** property **false**. **Parent** property is always recommended to be set so that when the parent control is disposed, **AudioInput** instance gets disposed automatically. If there is no parent, then **Dispose** method should be called when **AudioInput** instance is no longer needed. If a new **AudioInput** instance is created via Visual Studio's toolbox, **Parent** property is automatically set.

It is recommended to set **LicenseKey** property so that the **AudioInput** instance uses an explicit license key instead of trying to find one from Windows' registry. If a trial version/license is used, **LicenseKey** property can be left to its default value.

19.4.2 Event handling

To get new samples from **AudioInput** instance, the user needs to subscribe at least to **DataGenerated** event. When **DataGenerated** event is triggered, new samples and the first sample time stamp from a **DataGeneratedEventArgs** object are provided as a parameter.

Subscribe to **Started** event to know when **AudioInput** instance has started its audio sampling task. A **StartedEventArgs** object provides information about **AudioInput** as a parameter, for example the number of bits per sample, is the stream audio mono or stereo, and how many samples per second are generated.

Subscribe to **Stopped** event to know when **AudioInput** instance has stopped. The event has no parameters and its sole purpose is to tell user when everything has been stopped.

19.4.3 Configuring

Set **ThreadInvoking = true** to allow an **AudioInput** instance to synchronize its events to the main UI thread automatically but make sure that the **AudioInput** instance has a valid parent control. **ThreadInvoking** is set to **false** by default so do not forget to call **Control.Invoke** method if updating GUI in **DataGenerated** event handler.

Setting **RecordingDevice** property allows using other Windows' recording device than the default one. Get all available recording devices by using **AudioInput**'s static method **GetRecordingDevices**.

Volume can be controlled via **Volume** property. Valid values are from 0 to 100 where 0 means mute and 100 maximum volume. The volume can also be set when **AudioInput** instance is enabled (i.e. generating samples).

Set **SamplesPerSecond** property to use difference sampling rate than the default (44100 Hz). Setting this property while **AudioInput** instance is enabled has no effect.

To use mono audio instead of stereo (default), set **IsStereo** to **false**. Setting this property while **AudioInput** instance is enabled has no effect.

If 8 bits per sample is preferred rather than 16 (default), set **BitsPerSample** property to 8. Valid values are 8 and 16 (default). This limitation comes from PCM wave format. Setting this property while **AudioInput** instance is enabled has no effect.

19.4.4 Starting

To start **AudioInput** instance, either set **IsInputEnabled** property **true**, or call **Start** method. **DataGenerated** event then provides a new set of audio samples which can e.g. be rendered using **LightningChart Ultimate** instance.

19.4.5 Stopping

To stop **AudioInput** instance, set **IsInputEnabled** to **false**, or call **RequestStop** method. **RequestStop** method does not stop instantly. Instead, it signals **AudioInput** instance to stop as soon as it is possible. Subscribe to **Stopped** event to know when **AudioInput** instance has stopped.

19.5 Usage (WPF)

This chapter describes the usage of WPF version of **AudioInput** class. WPF version of **AudioInput** works mostly the same way as WinForms version. However, there are a couple of things that a WPF user should be aware of.

19.5.1 Creation

Create a new **AudioInput** instance either manually in code-behind or by dragging and dropping it from Visual Studio's toolbox on to a window, user control etc.

If there is no need to show GUI (i.e. if an own GUI or **AudioInput** object is controlled from the source code) then use **AudioInput** from **Arction.WPF.SignalTools** namespace. This particular class is derived from FrameworkElement and all its properties are bindable. For convenience, after having installed LightningChart Ultimate SDK, **Arction.WPF.SignalTools.AudioInput** can also be found from Visual Studio's toolbox so it can be dropped to a window, user control etc. and then moved in the XAML code to wherever it is needed. Necessary XML namespace will be added automatically this way.

There is also a ready-made GUI for **AudioInput**. It can be found in **Arction.WPF.SignalTools.GUI** namespace. Visual Studio's toolbox also has it after LightningChart Ultimate SDK has been installed. Note that this is just a GUI for **Arction.WPF.SignalTools.AudioInput** class but it contains an instance of **Arction.WPF.SignalTools.AudioInput** class which can be accessed **Input** property. In other words, there is no need to create a new separate **Arction.WPF.SignalTools.AudioInput** instance.

It is recommended to set **LicenseKey** property so that the **AudioInput** instance uses an explicit license key instead of trying to find one from Windows' registry. When using a trial version/license, **LicenseKey** property can be left to its default value.

20. AudioOutput component

AudioOutput component allows user to convert System.Double signal data into an audio stream which is then played back through speakers or sent to Line-out connector of sound device.

20.1 Properties

Balance – Gets or sets audio playback balance. Valid values are between -100 to 100. -100 means that audio is played only through the left speaker. 0 means that both speakers output audio. 100 means that audio is played only through the right speaker.

BitsPerSample – Gets or sets how many bits are allocated per sample. Supported values are 8 and 16. If any other value is used, 16 is used instead. It can be set when **IsOutputEnabled** is **false**.

IsOutputEnabled – Gets or sets the state of this instance (i.e. starts or stops it). Setting this property **true** is the same as calling **Start** method where **false** is the same as calling **Stop** method.

IsStereo – Gets or sets whether to use two channels (stereo) or just one (mono). It can be set when **IsOutputEnabled** is **false**.

LicenseKey – Gets or sets license key string in normal or encrypted format.

PlaybackDevice – Gets or sets the current playback device. Can be set when **IsOutputEnabled** is **false**. By setting this property **null**, Windows' default playback device is used.

SamplesPerSecond – Gets or sets sampling frequency. Can be set when **IsOutputEnabled** is **false**.

Volume – Gets or sets volume (0-100). Can be set when **IsOutputEnabled** is **false**.

21. SpectrumCalculator component

SpectrumCalculator component allows conversion between time domain and frequency domain.

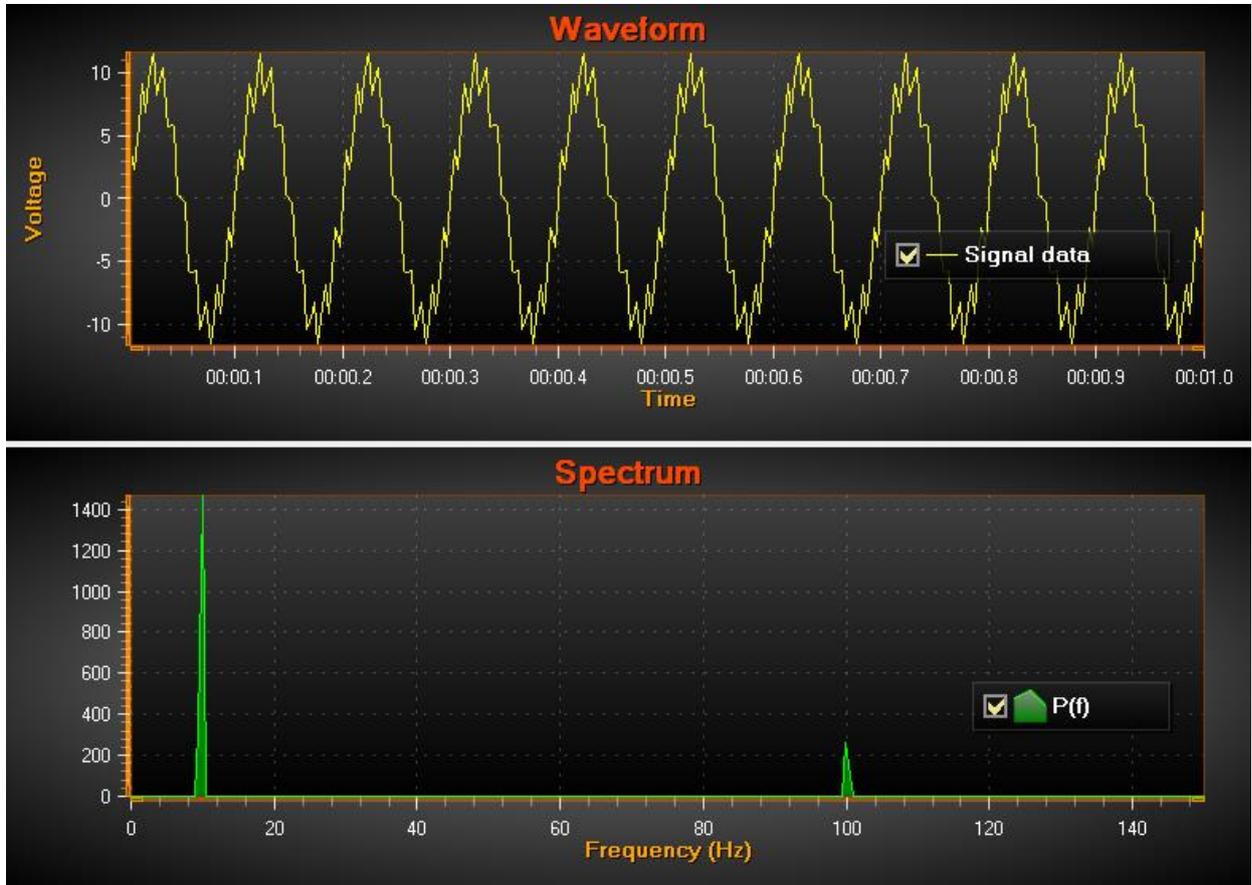


Figure 21-1. Example of source signal data (top) converted to frequency domain (bottom). Signal sampling frequency = 300 Hz, thus frequency scale is $300/2 = 150$ Hz. The strong sine base line is 10 Hz (10 cycles / sec). Smaller signal of 100 Hz is added as noise. Both spikes are found in the power spectrum.

The following public methods are available:

- **CalculateForward**(double[] samples, out double[] fftData) - Converts time domain signal data to frequency domain by using FFT. Output fftData contains also negative values. Input and output data arrays must be of equal length. The length is the resolution of the data, spreading from 0 Hz to sampling frequency / 2 with equal frequency interval between output values.
- **CalculateForward**(float[] samples, out float[] fftData) – Similar to the previous method, but for single accuracy floating point values.
- **CalculateBackward**(double[] fftData, out double[] samples) - Converts frequency domain data to time domain. Makes signal samples from FFT data. Sample count equals input fftData length.

- **CalculateBackward**(float[] fftData, out float[] samples) – Similar to the previous method, but for single accuracy floating point values.
- **PowerSpectrum**(double[] samples, out double[] fftData) - Calculates power spectrum of signal data. Is the same as **CalculateForward**, but with absolute output values.
- **PowerSpectrum**(float[] samples, out float[] fftData) – Similar to the previous method, but for single accuracy floating point values.
- **PowerSpectrumOverlapped**(double[] samples, int fftWindowLength, double overlapPercent, out double[] fftData, out int processedSampleCount) - Calculates the power spectrum by shifting the calculation windows inside source signal samples data, by overlap percent. Signal data must be longer than given FFT window length. The output FFT data is the length of fftWindowLength which is not necessarily the same as the length of the source data. The output data has absolute values.

22. Headless mode

Headless mode is a software capability of working on a device without access to Graphical User Interface (GUI). The term "headless" is also used when software does not require the presence of peripheral devices (like display, keyboard, mouse) or access to them. The absence of peripherals does not cause the failure of initialization or execution processes. However, in this case the software may receive inputs and provide output via other communication interfaces, for example via network or a serial port.

22.1.1 Headless Rendering

Headless configuration allows running LightningChart in a headless/server environment. Expected scenarios include background rendering in software applications without User Interface (UI) and generation of a bitmap image from the chart content. The image then can be passed to the headful system for further rendering.

Basic usage:

```
var chart = new LightningChartUltimate(new RenderingSettings()
{
    HeadlessMode = true
});
```

Headless mode can be activated by setting **HeadlessMode** flag to **true**. The property can be accessed via **chart.ChartRenderOptions** (for WPF) or **chart.RenderOptions** (for WinForms). LightningChart automatically detects its usage in the Windows Service type application, thus there is no need to specify the mode.

22.1.1.1 Additional initialization options

The initialized instance of LightningChart with a missing UI and visual parent will not receive any rendering requests, like sizing the layout or when to render a frame. Furthermore, WPF chart uses these signals to initialize a rendering engine, when WinForms does engine initialization during the creation time. Thus, the following operations and configurations must be applied to the chart by the user:

- Define size using **chart.Width** and **chart.Height** properties.
- Request rendering engine initialization by calling **chart.InitializeRenderingDevice(true)** (only for WPF).
- Subscribe to **chart.AfterRendering** event for implementing the logic of exporting the images.

The chart still reacts to property changes. The rendering of a new frame can be queried by consecutive **BeginUpdate()** and **EndUpdate()** call, if it is needed.

22.1.1.2 Capturing images

The rendered frame can be exported (see chapter 13) in various ways:

- **OutputStream** property
- **SaveToStream** method
- **CopyToClipboard** method
- **CaptureToByArray** method
- **SaveToFile** method

In general, bitmap stream is preferred. Also, ViewXY chart supports EMF, WMF, SVG in headless mode in **SaveToStream** and **SaveToFile** methods.

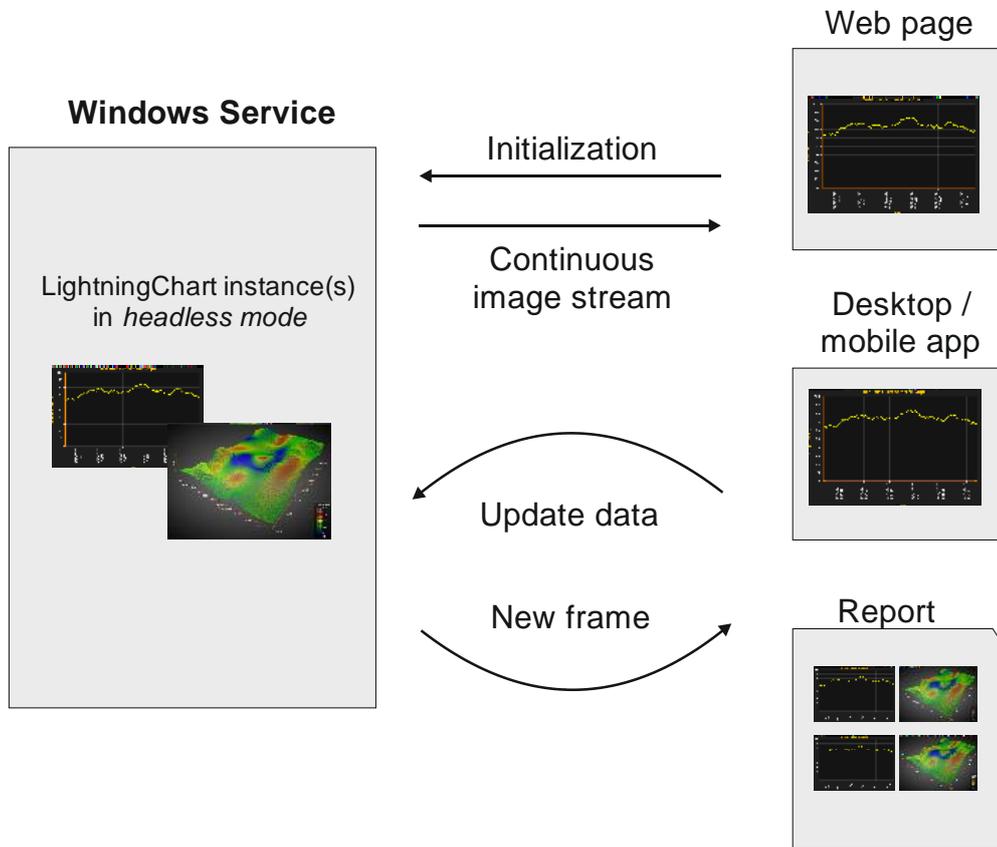


Figure 22-1. Diagram of example usages.

22.1.2 Limitations and Requirements

22.1.2.1 Threads

Headless configuration allows using LightningChart for a background work without placing it inside a visual parent and without access to the chart from the foreground thread (GUI thread). During the creation of the LightningChart instance, the properties of a chart must be updated within the same thread that has created the chart.

- The COM threading model, called “**apartment**”, must be STA (Single Thread Apartment), not MTA (Multi Thread Apartment). For an ordinary UI application, STA is the default model, whereas MTA state is default for Windows Services.
- All access, i.e. update, creation and disposing, must be made via that thread. UI must be touched only from GUI thread. Thus, if there are interaction operations required, they should be moved from chart’s thread to GUI thread. **Note!** LightningChart can be run on GUI thread.
- The thread must have a valid and active message queue pump. For example, run **Application.Run** on the thread.

22.1.2.2 Chart Update

LightningChart uses a single buffer on rendering, thus exporting of a new image will be handled after the exporting of the previous image is finished. The synchronous configuration (**ChartUpdateType.Sync**) provides a rendering of an image straight after receiving a request to update the properties of a chart. Sync mode should be enabled for the headless mode to enable faster and uninterrupted performance.

22.1.2.3 Engine support

Both DirectX 9 & 11 engines work in headless mode. However, only DirectX 11 can be used in Windows Services type applications due to the limitations of MS DirectX.

22.1.2.4 Licensing

By default, Windows Service executes in the security context of a system user account. **Installation of a trial and development license is impossible.** For this reason, the service application **must** contain a valid **Deployment Key** or be running with credentials of a normal user with an active license (trial / development).

22.1.3 Example solution

LightningChart SDK comes with an example Visual Studio solution (*DemoService.sln*) containing:

- Service
- Console application
- Client application for WPF

DemoService.sln can be found in *C:\ProgramData\Arction\LightningChart Ultimate SDK v.8\DemoService* folder.

When starting up the WPF client, it shows a frame container in the middle of the window.

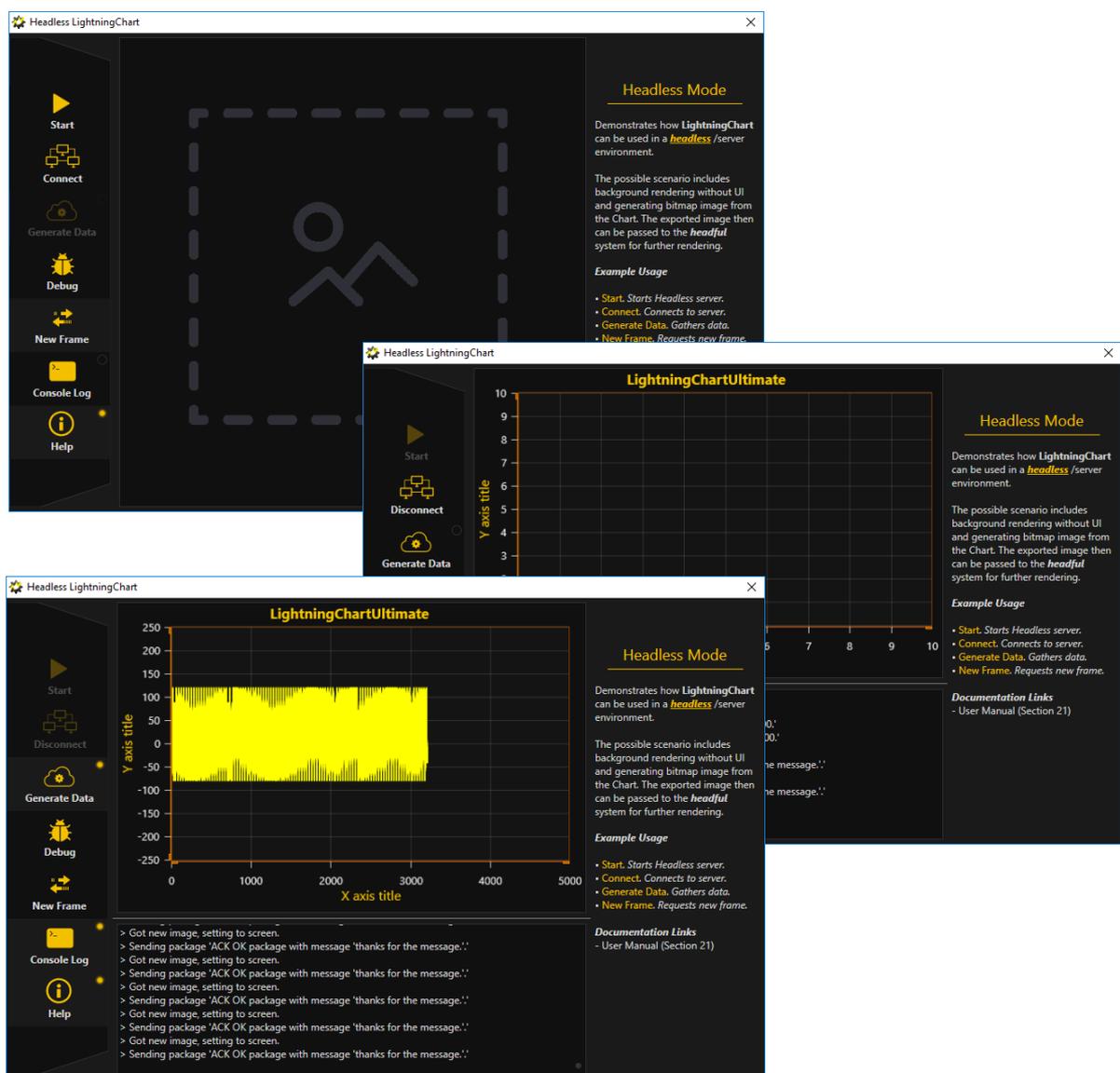


Figure 22-2. WPF client app. After Start, Connect and Generate Data, it shows continuously updating image stream.

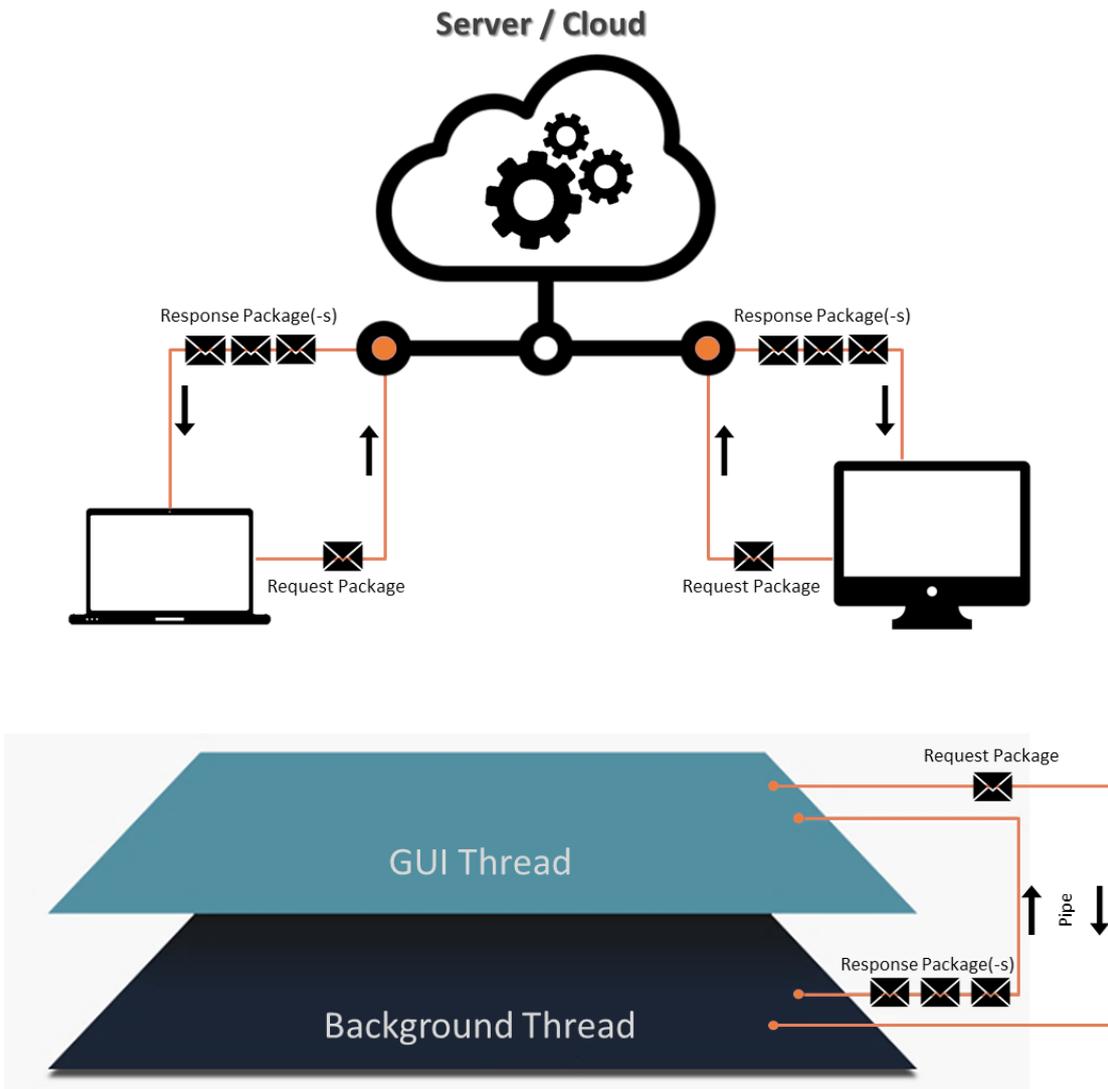


Figure 22-3. Headless demo service – client internal operation of messages with named pipes and background thread illustrated.

23. Using Windows Forms chart in WPF application

LightningChart has 3 WPF API's available. Consider using WinForms chart API in WPF application only in special cases.

23.1 How about using Arction Windows Forms controls in WPF?

In WPF, Windows Forms components can be used by adding **Arction.WinForms.Charting.LightningChartUltimate.dll** and **Arction.WinForms.SignalProcessing.SignalTools.dll** as reference to the project, and creating them by code. LightningChart Ultimate control and most of other controls have a built-in UI. Use **WindowsFormsHost** as parent container to these. These controls can be used also without UI, with their methods and properties.

23.2 Should I use Arction.WinForms.LightningChartUltimate in WPF?

Using WPF chart assemblies is recommended over WinForms chart in WPF applications, because it doesn't need the **WindowsFormsHost** control, and thus does not have the generic "airspace" problem of **WindowsFormsHost** control. Another advantage is that the WPF chart can have transparent background and the charts can be placed one over another.

Using **WindowsFormsHost** control with WinForms chart control can be considered to be used when the absolute maximum performance is required. **WindowsFormsHost** + WinForms chart rendering is slightly faster.

If the user chooses to use the WinForms chart in WPF application, it must be placed inside **WindowsFormsHost** control. Add a **WindowsFormsHost** control (found in the Visual Studio WPF Toolbox) into the WPF form.

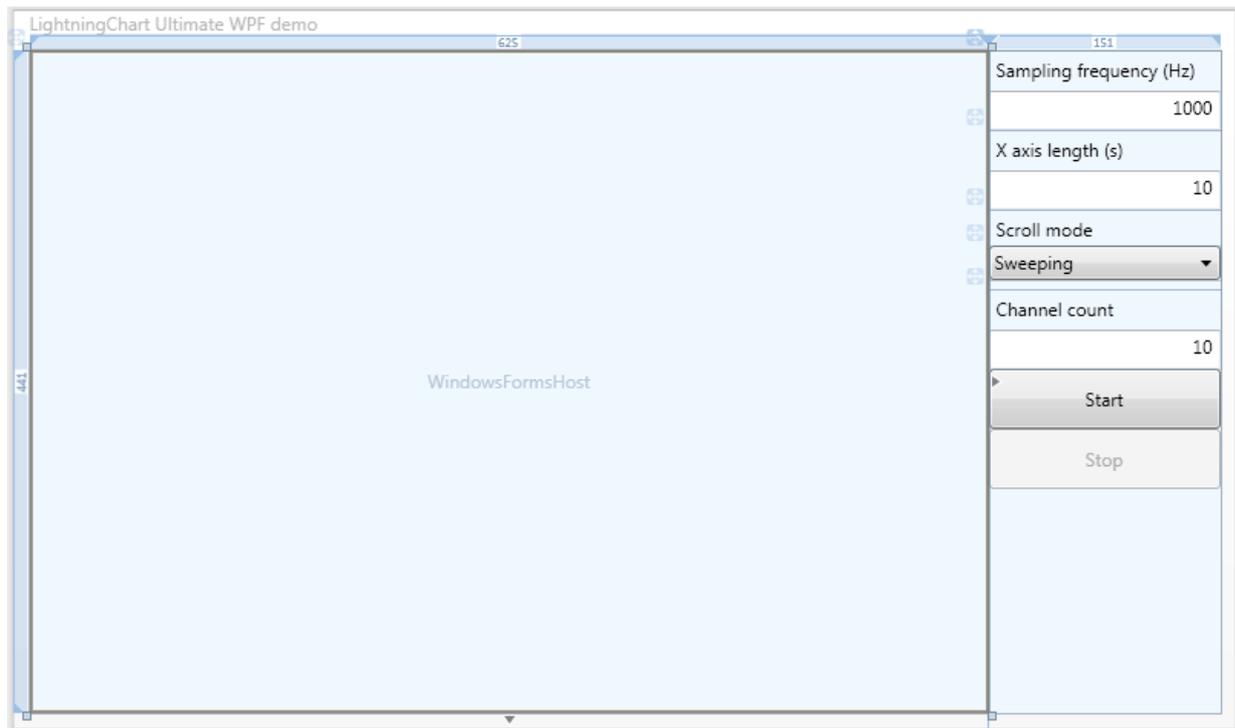


Figure 23-1. WPF application in designer. `WindowsFormsHost` control keeps the `LightningChart Ultimate` object inside when the application is executed.

Create a `LightningChart Ultimate` object and place it inside the `WinFormsHost` object in code. Open the form `xaml.cs` file and create the chart in the form constructor:

```
public WindowMain()
{
    InitializeComponent();

    CreateChart();
}

private LightningChartUltimate m_chart = null;

void CreateChart()
{
    m_chart = new LightningChartUltimate();

    //Set the chart object as child to the WindowsFormsHost control
    windowsFormsHost1.Child = m_chart;
}
```

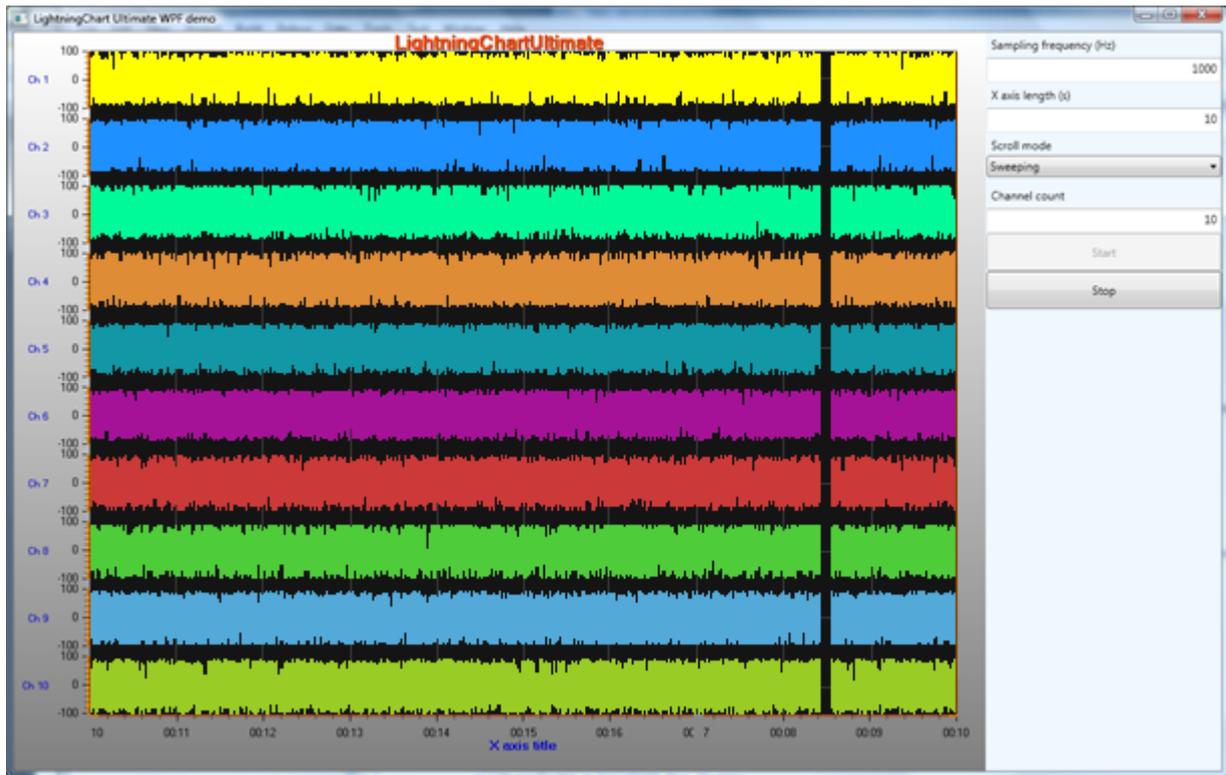


Figure 23-2. WinForms chart in a WPF application.

24. Using LightningChart in C++ applications

LightningChart is a .NET library which can be most fluently used with C# and VB.NET language. However, it is possible to use LightningChart in C++ Win32 applications as well, including MFC applications. The application using LightningChart must be compiled with **Common Language Runtime Support (/clr)** option. When creating a Windows Form Project using C++, refer to the detailed step by step tutorial below.

24.1 Install required C++/CLR packages

Make sure your Visual Studio have installed C++ package with C++/CLR. For example, run Visual Studio (2017) Installer, and select update/modify button. From Individual components select **C++/CLI support**. From Workloads select **Desktop development with C++**.

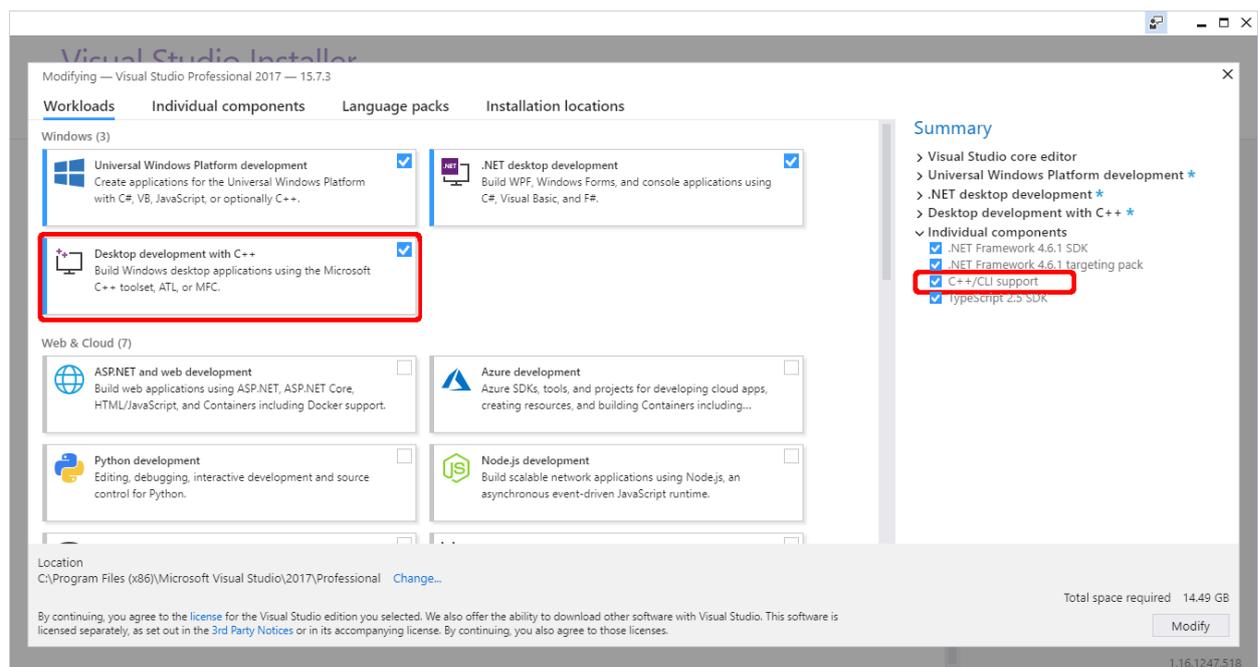


Figure 24-1. Visual Studio Installer selections for C++ project.

24.2 Setting Visual Studio project

Open Visual Studio 2017 and create a new project. If all the required packages and components described above are installed, the following selection is available when creating new project (**Templates -> Visual C++ -> CLR -> CLR Empty project**).

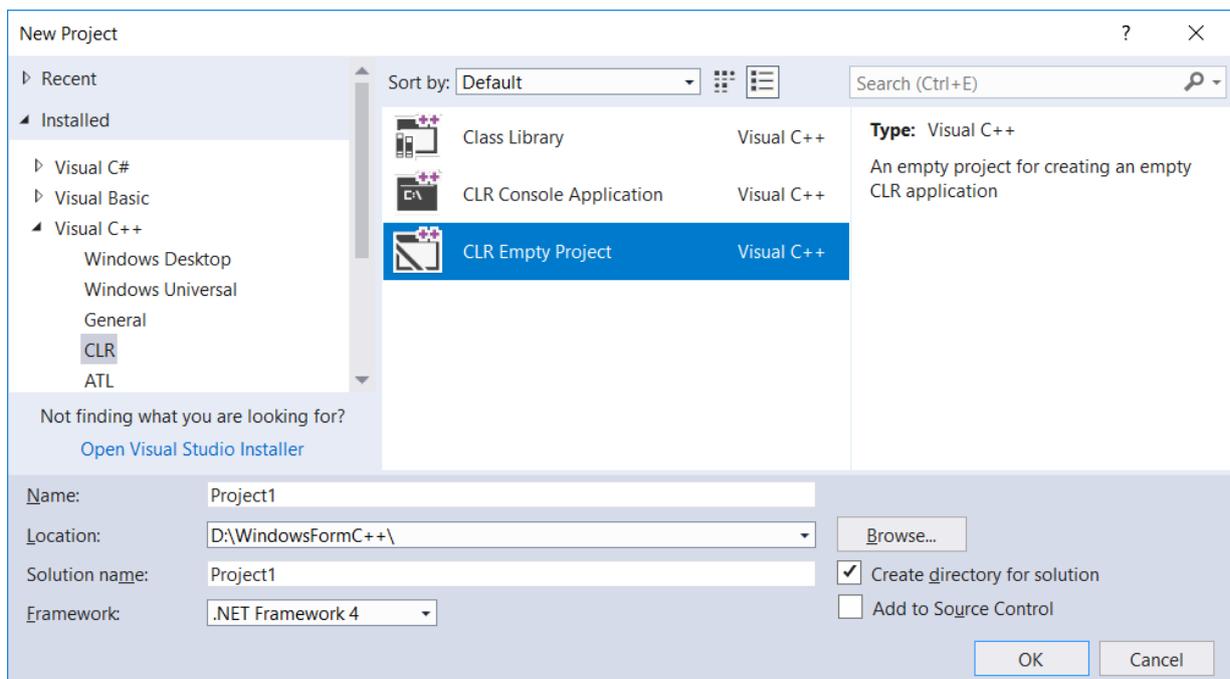


Figure 24-2. Windows Forms C++ project template.

Right click the created project and choose Properties option. Modify **Configuration Properties -> Linker -> System -> SubSystem** and **Linker -> Advanced -> Entry point** as show in the figure 24-3.

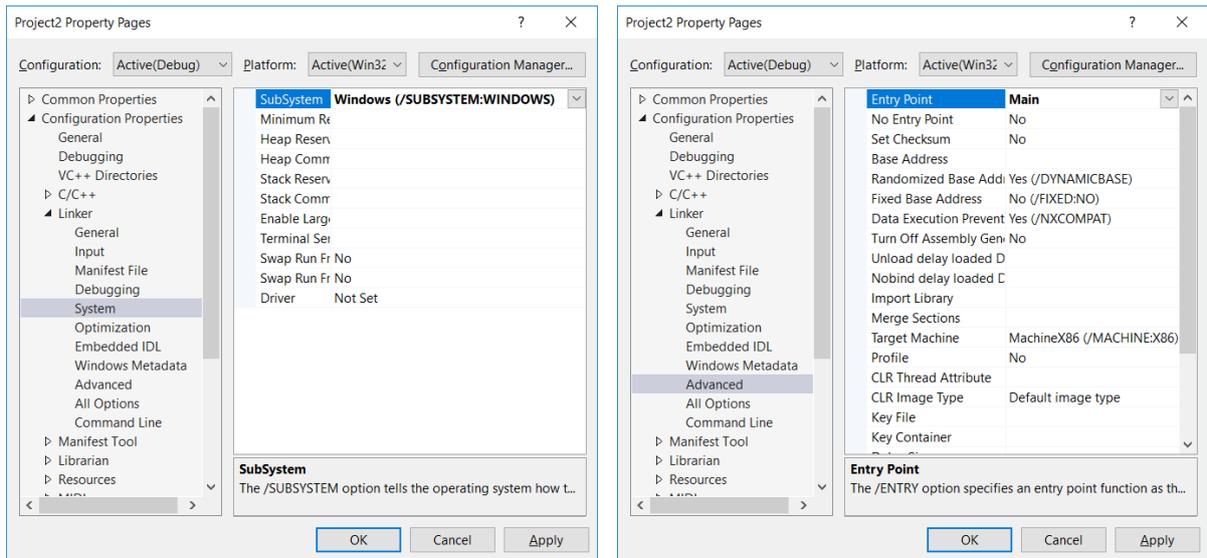


Figure 24-3. C++ project property pages.

Add *Windows Form* Item to the project: right click the project and choose **Add -> New Item...** Select *Windows Form* as shown in the figure 24-4. It is possible to get an error message: **The data necessary to complete this operation is not yet available. (Exception from HRESULT: 0x8000000A)**. This can be ignored, just close it and proceed to the next step.

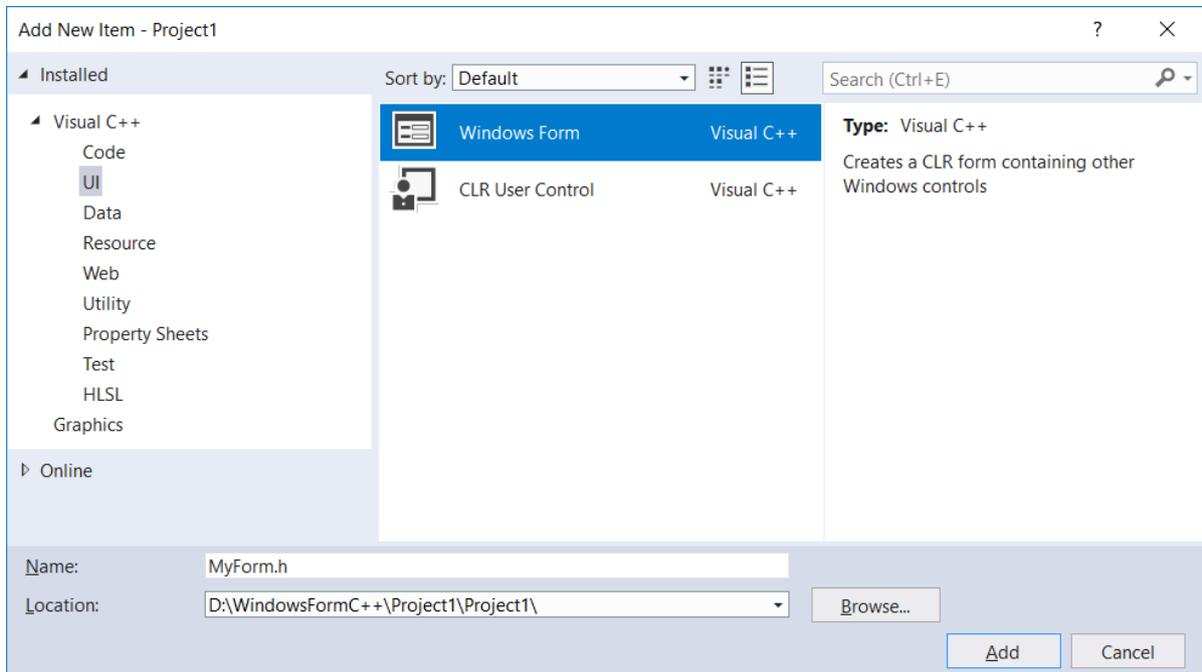


Figure 24-4. Add new Windows Forms item in C++ project.

Add the following code to the created form, save it and close the Visual Studio 2017.

```
#include "MyForm.h"

using namespace System;
using namespace System::Windows::Forms;

[STAThreadAttribute]
void Main(array<String>^ args) {
    Application::EnableVisualStyles();
    Application::SetCompatibleTextRenderingDefault(false);
    Project1::MyForm form;
    Application::Run(%form);
}
```

The project is ready to be built for the first time. Reopen the project and select **Build -> Rebuild Solution**. When the project is running, an empty *Windows Form* should be seen.

24.3 Creating LightningChart application in C++ project

Components can now be added to the form by editing **MyForm.h** file. Below is a simple example how to create a chart with **PointLineSeries** in it. Include Arction's WinForms DLL in references list and add relevant namespaces in **MyForm.h** code file.

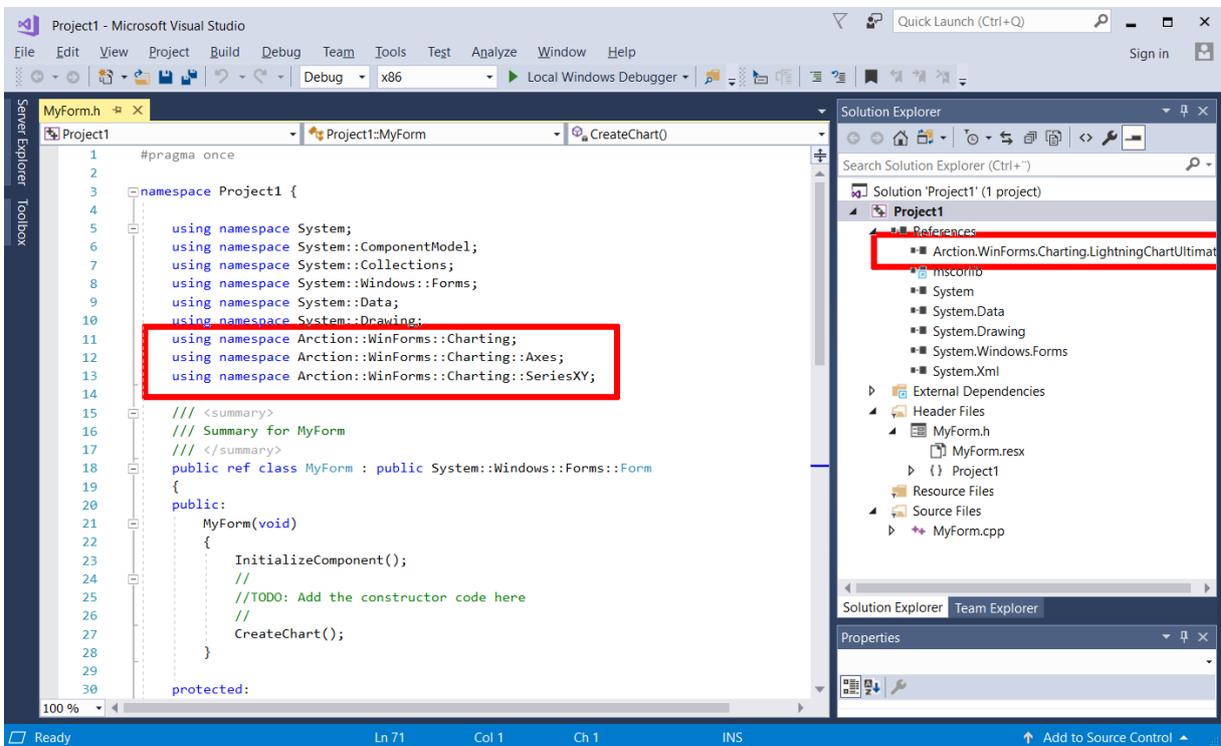


Figure 24-5. Include `Arction.WinForms.Charting.LightningChartUltimate.dll` in references list and add relevant namespaces in the project.

Declare a 'chart' variable and set its properties. Below is an example of a chart creation method.

```
protected:
LightningChartUltimate ^ _chart;

void CreateChart()
{
    _chart = gcnew LightningChartUltimate();

    //Disable repaints for every property change
    _chart->BeginUpdate();

    //Set parent window by window handle
    _chart->Parent = this;

    //Fill the form area
    _chart->Dock = DockStyle::Fill;

    _chart->ActiveView = ActiveView::ViewXY;

    // Configure x-axis.
    AxisX^ axisX = _chart->ViewXY->XAxes[0];
    axisX->SetRange(0, 20);
    axisX->ScrollMode = XAxisScrollMode::None;
    axisX->ValueType = AxisValueType::Number;

    // Configure y-axis.
    AxisY^ axisY = _chart->ViewXY->YAxes[0];
    axisY->SetRange(0, 100);

    PointLineSeries^ pls1 = gcnew PointLineSeries(_chart->ViewXY, axisX, axisY);
    pls1->LineStyle->Color = Color::Yellow;
    pls1->Title->Text = "New Title";
    pls1->PointsVisible = true;
    pls1->LineVisible = true;
    _chart->ViewXY->PointLineSeries->Add(pls1);

    // Generate random data.
    Random rand;
    int pointCount = 21;

    array<SeriesPoint> ^ points = gcnew array<SeriesPoint>(pointCount);
    for (int point = 0; point < pointCount; point++)
    {
        points[point].X = (double)point;
        points[point].Y = 100.0 * rand.NextDouble();
    }

    pls1->Points = points;

    // Allow chart rendering.
    _chart->EndUpdate();
}
```

The resulting application when compiled and executed:

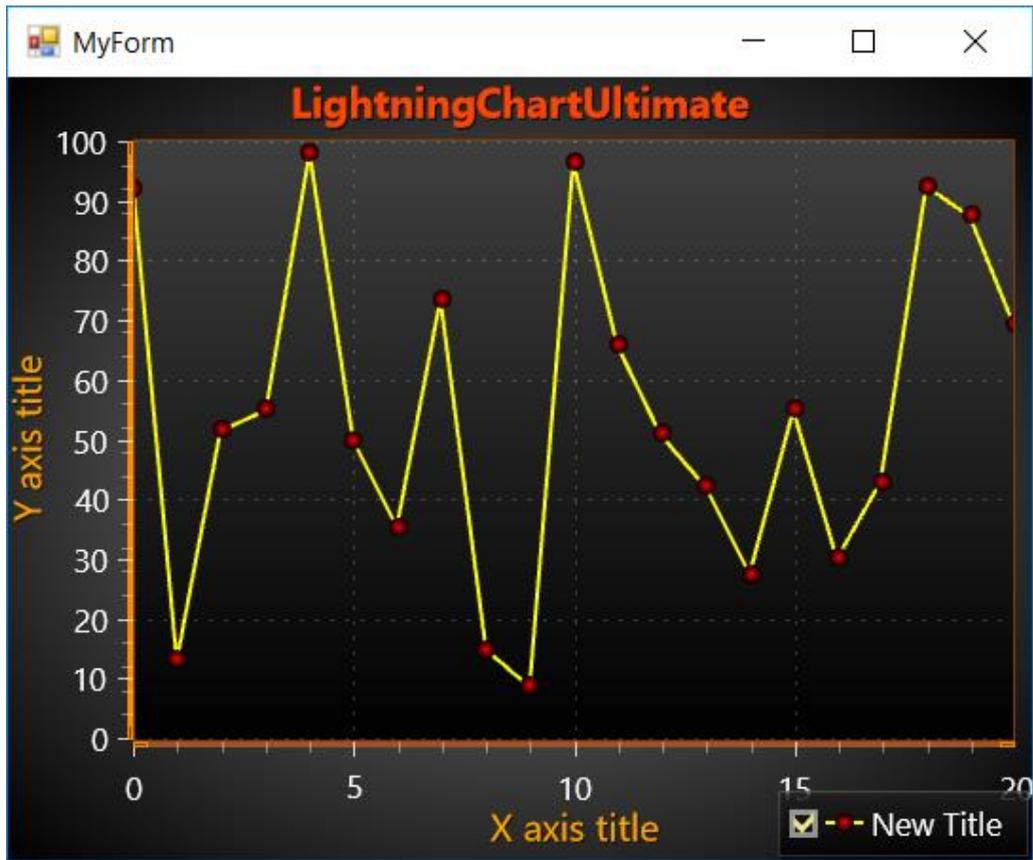


Figure 24-6. Example application executed.

25. Dispose pattern

25.1 Chart disposing

When a chart has been created in code, and is no longer needed, ***chart.Dispose()*** should be called. It frees the chart and all its objects, such as series, markers and palette steps from the memory.

25.2 Disposing objects

If objects are created on the fly, and then the memory used by them needs to be freed before exiting the application or disposing the whole chart with ***chart.Dispose()***, remove the object from the collection it has been added to, and then call ***Dispose()*** for the object.

For example, disposing a series from ***chart.ViewXY.PointLineSeries*** collection:

```
//Do cleanup... Remove and dispose 3 series

_chart.BeginUpdate();

List<PointLineSeries> listSeriesToBeRemoved = new List<PointLineSeries>();
listSeriesToBeRemoved.Add(_chart.ViewXY.PointLineSeries[1]);
listSeriesToBeRemoved.Add(_chart.ViewXY.PointLineSeries[3]);
listSeriesToBeRemoved.Add(_chart.ViewXY.PointLineSeries[4]);

foreach (PointLineSeries pls in listSeriesToBeRemoved)
{
    _chart.ViewXY.PointLineSeries.Remove(pls);
    pls.Dispose();
}

_chart.EndUpdate();
```

When LightingChart's objects are no longer needed, it is a good practice to dispose them to prevent memory leaking.

26. Object model notes

26.1 Sharing objects between other objects

LightningChart object model is tree-based. Every class has its parent object and a list of child objects. This tree-model allows child object to notify the parent object of its changes, allowing the parent to respond to it. Respectively, the parent notifies its parent and so on until the root node, LightningChart Ultimate itself, is reached, which then knows how to refresh accordingly.

Chart takes ownership of all the objects given to it and will dispose the objects when it no longer needs them. This includes situations where a new object replacing an old one is given to chart, and the parent of the object is disposed. User must be aware of this, as otherwise it is possible to end up using disposed objects.

If an object is shared between another .NET component and LightningChart, and LightningChart disposes the object, the .NET component is left with a disposed object. LightningChart cannot detect parent sharing between LightningChart and other components.

Sharing objects between other objects in the same chart, or other chart instances, is not allowed.

Example 1 of wrong usage:

```
AnnotationXY annotation1 = new Annotation();  
chart.ViewXY.Annotations.Add(annotation1);
```

```
AnnotationXY annotation2 = new Annotation();  
annotation2.Fill = annotation1.Fill;  
chart.ViewXY.Annotations.Add(annotation2);
```

Issue: The same **Fill** object cannot be shared between multiple objects.

Correct way: Only copy properties if they are of ValueType (e.g. Integer, Double, Color)

Example 2 of wrong usage:

```
SeriesEventMarker marker = new SeriesEventMarker();
```

```
chart.ViewXY.PointLineSeries[0].SeriesEventMarkers.Add(marker);
```

```
chart.ViewXY.PointLineSeries[1].SeriesEventMarkers.Add(marker);
```

Issue: The same object shouldn't be added to a collection of multiple collections.

Correct way: Create several markers, one for each series.

Remember to subscribe to ChartMessage event handler. In most cases it reports errors of invalid object sharing cases (see chapter 15).

27. Deployment / distribution of LightningChart assemblies

27.1 Referenced assemblies

Deliver Arction dlls with the executable folder, next to the executable folder, with Global assembly cache, or with another folder where .NET assembly resolving system can find them. LightningChart also supports **ClickOnce** deployment.

WinForms:

- Arction.WinForms.Charting.LightningChartUltimate.dll
- Arction.Licensing.dll
- Arction.DirectX.dll
- Arction.RenderingDefinitions.dll
- Arction.RenderingEngine.dll
- Arction.RenderingEngine9.dll
- Arction.RenderingEngine11.dll
- Arction.DirectXInit.dll
- Arction.DirectXFiles.dll

If using SignalTools

- Arction.WinForms.SignalProcessing.SignalTools.dll
- Arction.MathCore.dll

WPF:

- Arction.Wpf.Charting.LightningChartUltimate.dll (**for Non-bindable WPF chart**)
- Arction.Wpf.SemibindableCharting.LightningChartUltimate.dll (**for semi-bindable WPF chart**)
- Arction.Wpf.BindableCharting.LightningChartUltimate.dll (**for fully bindable WPF chart**)
- Arction.Licensing.dll
- Arction.DirectX.dll
- Arction.RenderingDefinitions.dll
- Arction.RenderingEngine.dll
- Arction.RenderingEngine9.dll
- Arction.RenderingEngine11.dll
- Arction.DirectXInit.dll
- Arction.DirectXFiles.dll

If using SignalTools

- Arction.Wpf.SignalProcessing.SignalTools.dll
- Arction.MathCore.dll

27.2 License key

Remember to use static **SetDeploymentKey** method for all components. Otherwise the chart enters in trial mode and works only for 30 days, with a trial nag on it. *For making a DeploymentKey and detailed license keys management, see chapter 3.*

27.3 Obfuscating application code

It is **mandatory** to obfuscate the application code, so that LightningChart license keys are not visible to .NET disassembler tools. Leaking license keys may lead into license termination, legal actions and claim of damage.

27.4 Obfuscating LightningChart code

A LightningChart source code subscriber gets access to the source code of LightningChart libraries. It is **mandatory** to obfuscate the assemblies build from LightningChart source code, to prevent Arction's intellectual property rights and code from leaking. Distributing unobfuscated LightningChart libraries is a violation of EULA, and may lead into license termination, legal actions and claim of damage.

27.5 XML files of Arction assemblies

Deployment of these XML files is forbidden.

- Arction.WinForms.Charting.LightningChartUltimate.xml
- Arction.Wpf.BindableCharting.LightningChartUltimate.xml
- Arction.Wpf.Charting.LightningChartUltimate.xml
- Arction.Wpf.SemibindableCharting.LightningChartUltimate.xml
- Arction.Wpf.SignalProcessing.SignalTools.xml
- Arction.WinForms.SignalProcessing.SignalTools.xml

The files provided by Arction are only for helping with the application development. They are used mainly to show code parameters and property tips. When rebuilding LightningChart assemblies from source code, ensure the XML files mentioned above are not deployed. **Distributing them is strictly forbidden**, as they will reveal too much info for .NET disassembler and reverse-engineering applications.

28. Troubleshooting

28.1 Updating from older version

LightningChart components API may have been changed from an older version, after which the project may not load or use the new version automatically. These instructions show how to set the new version assemblies as a reference to a project and how to fix the properties that were unable to de-serialize in the Visual Studio form editor.

In order to update chart, the reference to the old version has to be removed and a reference to the new version added. In some cases *.Designer.cs and *.resx files may need to be fixed as they may contain properties, which are binary incompatible.

Removing the old reference from project References

1. Go to Solution Explorer.
2. Open References folder.
3. Select Arction assemblies and remove them by pressing **Delete** button or right-click and select **Remove**.

Adding a reference to a new version

1. Go to Solution Explorer.
2. Open References folder.
3. Add reference to new chart. Right-click on References folder. Select **Add Reference...** and select the new Arction DLL file.

As the API may have been changed, the source code on changed properties may have to be fixed.

If a chart is totally incompatible (i.e. Visual Studio can't load UI on form editor), LightningChart property setters have to be removed from *.Designer.cs and *.resx files.

Removing property setters from *.Designer.cs file

1. Open *.Designer.cs file in text editor (use other editor than Visual Studio if possible).
2. Locate and delete rows containing setters for LightningChart Ultimate. For example:
this.m_chart.Background =
(Arction.LightningChartUltimate.Fill)(resources.GetObject("m_chart.Background"));

There is no need to remove inherited properties, like Location and Size. Remove properties, which are read from resource by a method like above "NN = ((...)(resources.GetObject("...")));".

Removing serialized items from *.resx file

1. Open *.resx file in text editor.
2. Find xml tags containing Arction objects (they are identified with chart member name. e.g. "m_chart" or "lightningChartUltimate1").
3. Remove the lines including <data> tag to the end of xml object (</data> tag).

E.g. if chart background is serialized as the following xml object, all the following lines should be removed from the *.resx file:

```
<data name="m_chart.Background" mimetype="application/x-
microsoft.net.object.binary.base64">
<value>
    AAEEAAD/////AQAAAAAAAAAMAgAAAGRbCmN0aW9uLkxpczZ2h0bmluZ0NoYXJ0VWx0aW1hd
    GUsIFZlcnNp
    b249NC42LjEuMjAwMSwgQ3VsdHVyZT1uZXV0cmFsLCBQdWJsaWNlZX1Ub2t1bj03MmY1N
    WZiZDY5MDFm
    ... lots of encoded stuff ...
    YXlvdXBAAAAB3ZhbHVlX18ACAIAAAAAAAAAACw==
</value>
</data>
```

Note that some objects may be very large, e.g. title row count may be approximately 200 lines while views are usually much larger (View3D has about 2000 lines).

In case of having several charts, all their serialized properties need to be removed. Editor search is a handy tool to find the chart objects.

After the objects are removed from *.resx file and related setters from *.Designer.cs file, it should be possible to open the project successfully in Visual Studio form editor.

28.2 Web support

See www.arction.com/support for support options.

Discussion forums are available at <http://www.arction.com/forum>

28.3 Running in Virtual Machine platforms

LightningChart comes with DirectX10/11 WARP rendering for systems that don't give access to graphics hardware. Since WARP rendering takes place in CPU, performance reduction is to be expected when compared to hardware rendering. This needs an operating system with support for DirectX11.

For systems that don't support DirectX11, LightningChart falls back to DirectX9 Reference Rasterizer mode. Performance is very poor, only small fraction of WARP's performance. For automatic fallback to WARP and DirectX9, keep the RenderDevice set to **Auto**, **AutoPreferD9** or **AutoPreferD11** (chapter 4.8).

29. Credits

29.1 Intel Math Kernel library

LightningChart Ultimate SDK uses Intel Math Kernel Library in some parts, for example Fast Fourier Transform methods. Arction assemblies contain some native DLL files built from this library. Arction Ltd. is licensed to use Intel Math Kernel Library.

29.2 Open-source projects

We present thanks to the following open-source projects and material providers:

DirectX library for .NET

LightningChart uses SharpDX-derived DirectX .NET DLLs with Arction-made extensions, <http://www.sharpx.org/>

Map sources

LightningChart Ultimate maps have been imported from the map providers as follows:

<i>World, North America, Europe:</i>	Natural Earth, http://www.naturalearthdata.com/
<i>Australia:</i>	Australian Bureau of Statistics, http://www.abs.gov.au/
<i>Roads of USA:</i>	National Atlas of the United States, http://www.nationalatlas.gov

Scalable Vector Graphics output

LightningChart SVG export is using partially SvgNet project code by RiskCare Ltd.

Polynomial regression

Polynomial regression calculation code is partially based on Math.Net library, <http://www.mathdotnet.com/>

The modified source code parts are available per request free-of-charge from Arction Support (support@arction.com).

For copyrights notices of open-source projects, see **Readme for LightningChartUltimate.txt** in LightningChart SDK install folder.