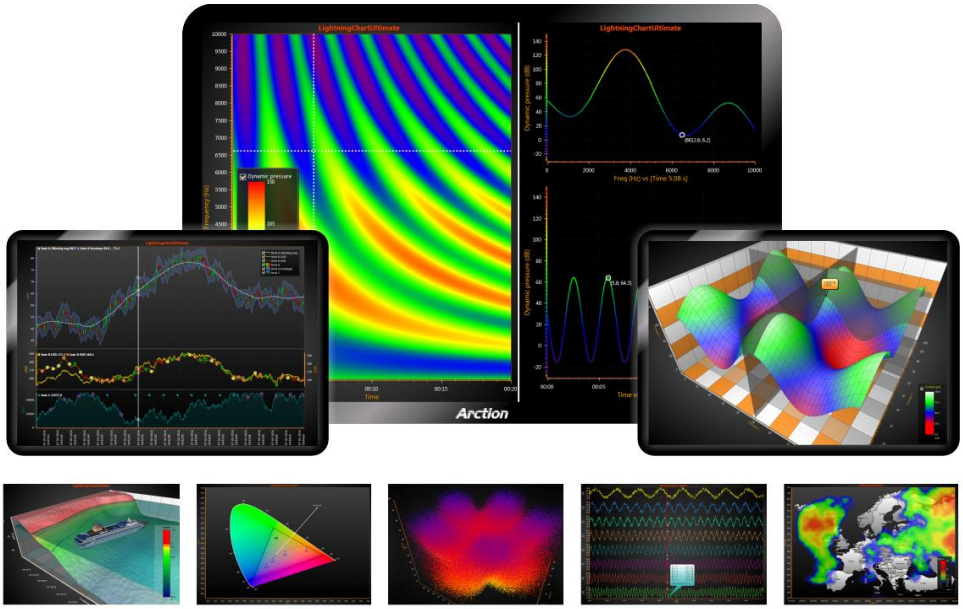


LightningChart®

使用手册

简介

本文档是简要的 **LightningChart®.NET** 使用手册及参考指南。本文仅对其中一些基本的关键特征进行了介绍，并没有对上百种类、属性或方法进行逐个说明。运行所提供的演示应用程序，可以快速预览一些 LightningChart 功能。有关如何通过代码使用 LightningChart 组件，请查览包含源代码的演示应用程序。

本文档中的所有代码示例均以 C#语言编写。大部分演示应用程序也提供有 C#和 Visual Basic.NET 的代码预览。

使用手册的其他语言版本，可访问 LightningChart .NET Resources 网站获取：

(https://www.lightningchart.com/lightningchart-net-resources/).

如果您有任何问题，请随时与我支持部门联系（support@lightningchart.com）！

适用于 LightningChart .NET，V.12.0

www.lightningchart.com

目录

1. 概述

LightningChart®.NET SDK 是一款 Microsoft Visual Studio 插件工具，由数据可视化软件组件和工具类组成，可支持基于 Windows 的用户界面框架（*Windows Presentation Foundation*）、Windows 通用应用平台（*Universal Windows Platform*）和 *Windows Forms*.NET 平台。

Arction 组件主要用于为科学、工程、测量和交易等领域提供解决方案，特别专注于执行性能和一些非常高级的功能。

LightningChart 组件摈弃了较慢的 GDI/GDI+或 WPF Graphics API，采用低版本的 DirectX9 和 DirectX11 GPU 加速。当 GPU 无法访问时，例如在某些虚拟机平台，LightningChart 可自动应变启用 DirectX11/DirectX10 WARP 软件渲染。

1.1 图表版本

对于 WPF，LightningChart 组件可用于各种绑定级别版本，以平衡不同的性能和 MVVM（模型 - 视图 - ViewModel）可绑定性需求。 可绑定图表 API 与不可绑定 WPF 图表类似，但附带扩展属性绑定，该绑定还涵盖在集合中创建的对象。 UWP 图表基于可绑定的 WPF 版本，提供类似的性能、绑定和 MVVM 功能。

图表版本	属性绑定	系列数据绑定	单个数据点绑定	性能
WPF （无绑定）	否	否	否	优秀
WPF （全绑定）	是	是	否	很好
UWP （全绑定）	是	是	否	很好
WinForms	否	否	否	最佳

表 1-1. 可绑定性与性能矩阵。

- 要在 WPF 中达到最佳性能以及多线程优势，建议选用无绑定图表。
- 要在 WPF 可绑定性和性能之间做好权衡，建议选用全绑定图表。全绑定图表还支持 MVVM 设计模式。

全绑定图表 API 与 LightningChart v.6 的 WPF 图表非常相似，只是附带有扩展属性绑定, 还包含有在集合中创建的对象。

在同一应用程序中可以使用不同的图表版本。因此，可以使用全绑定图表来创建基本图表，然后绑定各种属性，同时将无绑定图表用于高性能要求的任务。全绑定的集合属性（例如 ViewX 轴, 3D 光）默认为空，可完全支持 XAML 编辑器。而在无绑定和 WinForms 集合中则会预先填充默认项。

单个数据点绑定只在完全绑定的 WPF 中可支持，且仅以源代码方式实现：可构建源代码客户端。8.5 版是 Arction 官方支持的最后一个具有单个数据点绑定特性的版本。

注意，无绑定 WPF 图表无法在 XAML 里进行配置，可采用代码隐藏（代码后置）方式使用。

1.2 组件

标记有 **X** 的组件没有用户界面（UI）。

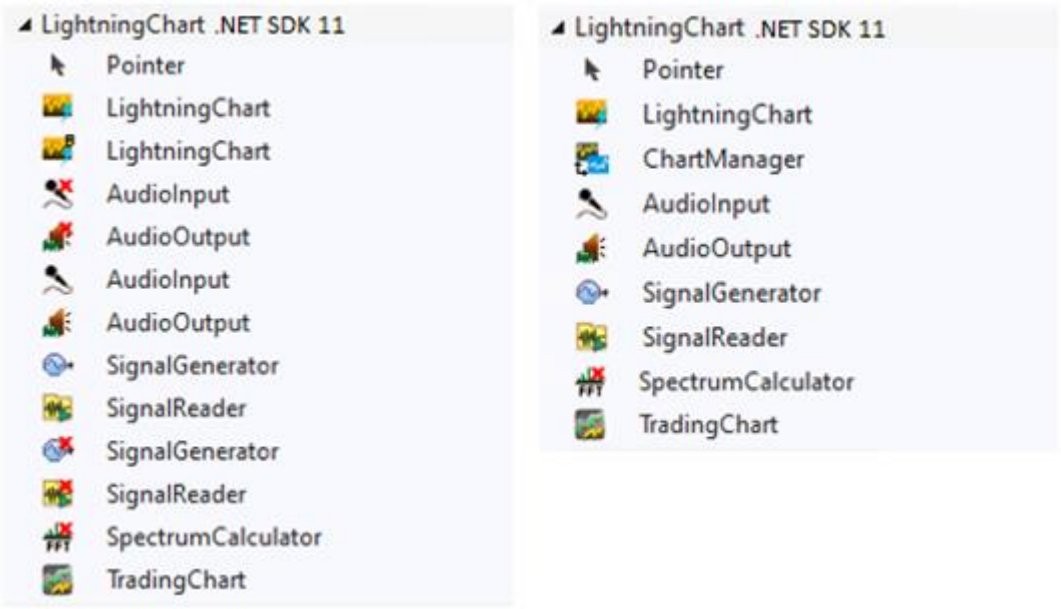


图 1-1. 左侧为 WPF 工具箱组件；右侧为 WinForms 工具箱组件

Charting 程序集

LightningChart： 图表组件。将各种展示内容的数据可视化。

图标上角， ***B = Bindable WPF 图表***

UWP chart, UWP 应用程序中可用。

ChartManager： 管控多个图表组件的互操作以及实时测量内存管理。详情请参阅第 17 章。

TradingCharts 程序集

TradingChart: 用于交易和金融类应用的图表控件。交易类库程序建立在 LightningChart API 之上。详情请参阅第 18 章。

SignalTools 程序集

标记有 **X** 的组件没有用户界面（UI）。

AudioInput （音频输入）从声音设备中读取波形音频流。声音设备中可用的标准选项为线路输入连接器或麦克风输入连接器。实时流可以转发到其他控件。详情请参阅第 211 章。

AudioOutput （音频输出）通过声音设备来播放实时数据流，例如：扬声器或线路输出。可以采取任何采样的实时信号，不一定是音频流。详情请参阅第 222 章。

SignalGenerator （信号生成）从多个可配置的波形组件中生成信号。详情请参阅第 189 章。

SignalReader（信号读取）从信号文件中读取波形数据，例如采用 PCM 编码的 WAV 格式文件。详情请参阅第 20 章。

SpectrumCalculator（频谱计算）利用 FFT（快速傅立叶变换）将信号数据（时域）转换为频谱（频域）。同时还包含由频域转到时域的逆向转换方法。详情请参阅第 233 章。

1.3 命名空间

Chart edition	Assembly name	Namespace root	XML namespace
WPF .NET4 (non-bindable)	LightningChart. **Wpf.Charting**.NET4.dll	LightningChartLib. WPF.Charting	xmlns:lcunb= "http://schemas.lightningchart .com/charting/"
WPF .NET6 (non-bindable)	LightningChart. **Wpf.Charting**.NET6.dll	LightningChartLib. WPF.Charting	xmlns:lcunb= "http://schemas.lightningchart .com/charting/"
WPF .NET4 (bindable)	LightningChart. **Wpf.ChartingMVVM**.NET4.dll	LightningChartLib. WPF.ChartingMVVM	xmlns:lcusb= "http://schemas.lightningchart .com/chartingMVVM/"
WPF .NET6 (bindable)	LightningChart. **Wpf.ChartingMVVM**.NET6.dll	LightningChartLib. WPF.ChartingMVVM	xmlns:lcusb= "http://schemas.lightningchart .com/chartingMVVM/"
UWP	LightningChart.	LightningChartLib. UWP.ChartingMVVM	xmlns:lcu= "using:LightningChartLib.UWP.

	UWP.ChartingMVVM.dll		chartingMVVM"
WinForms .NET4	LightningChart. **WinForms.Charting**.NET4.dll	LightningChartLib. WinForms.Charting	N/A
WinForms .NET6	LightningChart. **WinForms.Charting**.NET6.dll	LightningChartLib. WinForms.Charting	N/A

表 1-2. LightningChart®.NET 版本的程序集名称与命名空间。

UWP 在 XML 中使用多种命名空间，最常见的几种如下：

```
xmlns:lcu="using:Arction.Uwp.ChartingMVVM”
xmlns:viewxy="using:Arction.Uwp.ChartingMVVM.Views.ViewXY”
xmlns:axes="using:Arction.Uwp.ChartingMVVM.Axes”
xmlns:titles="using:Arction.Uwp.ChartingMVVM.Titles”
xmlns:seriesxy="using:Arction.Uwp.ChartingMVVM.SeriesXY”
```

若使用 ViewXY 之外的其他视图，则使用单独视图和系列名称（View3D, ViewPolar 等）。

在 UWP 中应用命名空间示例：

```
<lcu:LightningChart
            ChartName="Line and Bars Chart">
            <lcu:LightningChart.ViewXY>
                        <viewxy:ViewXY>
                                    <viewxy:ViewXY.XAxes>
                                                <axes:AxisX Maximum="10"/>
                                    </viewxy:ViewXY.XAxes>
                        </viewxy:ViewXY>
            </lcu:LightningChart.ViewXY>
</lcu:LightningChart>
```

2. 安装

2.1 安装前

检查计算机配置是否符合要求

• DirectX 9.0c（着色器模型 3）级图形适配器或更新版本，或 DirectX11 兼容操作系统，无需图形硬件即可进行渲染。 建议使用 DirectX11 兼容的图形硬件。

• Windows Vista、7、8、10 或 11（32 位或 64 位）、Windows Server 2008 R2 或更高版本

• Visual Studio 2010-2022 用于开发，部署不需要。 对于多个最新的 LightningChart 版本，由于使用 .NET Framework 4.6 或更高版本，因此至少需要 Visual Studio 2012。 .NET（5、6、7、8）项目需要 Visual Studio 2022。

• 安装.NET Framework v.4.0 或更高版本。 LightningChart v10.0.1 之前的版本面向 .NET Framework 4.0。 对于更高版本的 LightningChart，目标 .NET Framework 有

2.2 .NET 兼容性

LightningChart 主要为 .NET 框架构建，但也兼容以下 .NET 版本：

• .NET Core 3.0 和 3.1
• .NET 5
• .NET 6
• .NET 7
• .NET 8

使用上述内容时，Visual Studio 可能会发出有关使用不同目标的已安装包的警告，特别是在使用 NuGet 包时。 但是，这并不妨碍应用程序工作。
在这些情况下，可以抑制或忽略警告。

从版本 11.0 开始，LightningChart .NET6 程序集可以在 .NET 6 及更高版本的项目中使用，而不会出现兼容性问题。 为此，必须将 System.Drawing.Common 包安装到项目中。

请注意，LightningChart 不会出现在 .Net Core 3、.NET 5-8 的 Visual Studio 工具箱中项目。 因此，必须在代码中创建和配置图表。

2.3 运行安装向导

右键单击 **LightningChart .NET SDK v10.exe**。安装程序会将组件安装到 Visual Studio Toolbox（工具箱）中, 另外还会安装与工具箱控件相关联的帮助文件。如果组件或帮助文件安装失败，请按照以下各节中的说明进行手动安装。

在试用 LightningChart 时，很可能要用到 SetupDownloader.exe 来下载并安装 SDK，也就是说不需要运行 LightningChart .NET SDK v10.exe 程序。

2.4 将 Arction 组件手动添加至 Visual Studio Toolbox

WinForms

1. 打开 Visual Studio，创建一个新的 **WinForms** 项目。右键单击“Toolbox”，选择“**Add Tab**”，并命名为“Arction”
2. 右键单击“Arction”选项卡，然后选择“**Choose items**... ”
3. 在“**Choose Toolbox items**”窗口中，选择“**.NET Framework Components**”页。点击“**Browse**...”

从安装组件的文件夹中，一般为, ***C:\Program Files\LightningChart\LightningChart .NET SDK v.11\LibNET4***，搜索到 **Arction.WinForms.Charting.LightningChart.dll 和 Arction.WinForms.SignalProcessing.SignalTools.dll**，点击“Open”。 然后就可以在“Toolbox”中找到这些组件了。

WPF

1. 打开 Visual Studio，创建一个新的 **WPF** 项目。右键单击“Toolbox”，选择“**Add Tab**”，并命名为“Arction”
2. 右键单击“Arction”选项卡，然后选择“**Choose items**... ”
3. 在“**Choose Toolbox items**”窗口中，选择“**WPF Components**”页。点击“**Browse**...”

从安装组件的文件夹中，一般为 ***C:\Program Files （x86）\Arction\LightningChart .NET SDK v.11\LibNet4***，搜索到 **Arction.Wpf.Charting.LightningChart.dll、Arction.Wpf.ChartingMVVM.LightningChart.dll 和 Arction.Wpf.SignalProcessing.SignalTools.dll**，点击“Open”。 然后就可以在“Toolbox”中找到这些组件了。

2.5 手动配置 Visual Studio 2010-2022 帮助

本章介绍了如何手动安装 LightrnngChart Ultimate®.NET 帮助内容。如果 Visual Studio 2010-2019 没有安装任何本地帮助内容，需要用到本章信息。当安装 LightningChart Ultimate®.NET 时，如果没有安装任何本地帮助内容，则不会安装 LightningChart Ultimate®.NET 的帮助。

完成此次配置后，从 Visual Studio 2010-2019 中可以查看 LightningChart Ultimate®.NET 的帮助。可以在 LightningChart Ultimate 的类型、属性等部分按 F1，或使用 Microsoft Help Viewer（帮助查看器）来浏览帮助内容。

2.5.1 Visual Studio 2010

请按照以下步骤在 Visual Studio 2010 上手动安装 LightningChart®.NET 帮助内容：

1. 打开 Visual Studio 2010.
2. 选择 ***Help -> Manage Help Settings***.
3. 在 Help Library Manager 中, 单击 ***Settings*** 链接.
4. 确保 ***I want to use local help*** 为选中状态.
5. 如果已选择好 ***I want to use local help***, 单击 ***Cancel*** 返回到 Help Library Manager。否则单击 ***OK***。
6. 单击 ***Install content from disk*** 链接
7. 单击 ***Browse*** 按钮，找到安装 LightningChart®.NET 的文件夹，默认安装路径为 ***C:\Program Files （x86）\Arction\LightningChart .NET SDK v.10 \MSHelpViewer***。
8. 选择 ***HelpContentSetup.msha*** 然后单击 ***Open*** 按钮。
9. 单击 ***Next*** 按钮。
10. 单击 LightningChart®.NET Help 旁边的 ***Add*** 链接，确保 ***Status*** 列值变为 ***Update Pending***。
11. 单击 ***Update*** 按钮。如果 Help Library Manager 询问是否要继续，单击 ***Yes*** 按钮。帮助库开始更新。
12. 帮助库更新完成后，单击 ***Finish*** 按钮关闭 Help Library Manager。

2.5.2 Visual Studio 2012-2022

请按照以下步骤在 Visual Studio 2012-2019 上手动安装 LightningChart®.NET 帮助内容：

1. 打开 Visual Studio 2012、2013、2015 、2017 或 2019。
2. 选择 ***HELP -> Add and Remove Help Content***。
3. Microsoft Help Viewer 启动后，选择 ***Manage Content***。
4. 选择 ***Disk*** under ***Installation source***。
5. 单击三个点的按钮以浏览文件。
6. 找到安装 LightningChart®.NET 的文件夹，默认安装路径为 ***C:\Program Files （x86）\Arction\LightningChart .NET SDK v.10\MSHelpViewer***
7. 选择 ***HelpContentSetup.msha*** 然后单击 ***Open*** 按钮。
8. 单击 LightningChart®.NET Help 旁的 ***Add*** 连接，确保 ***Status*** *列值变为* ***Add pending***。

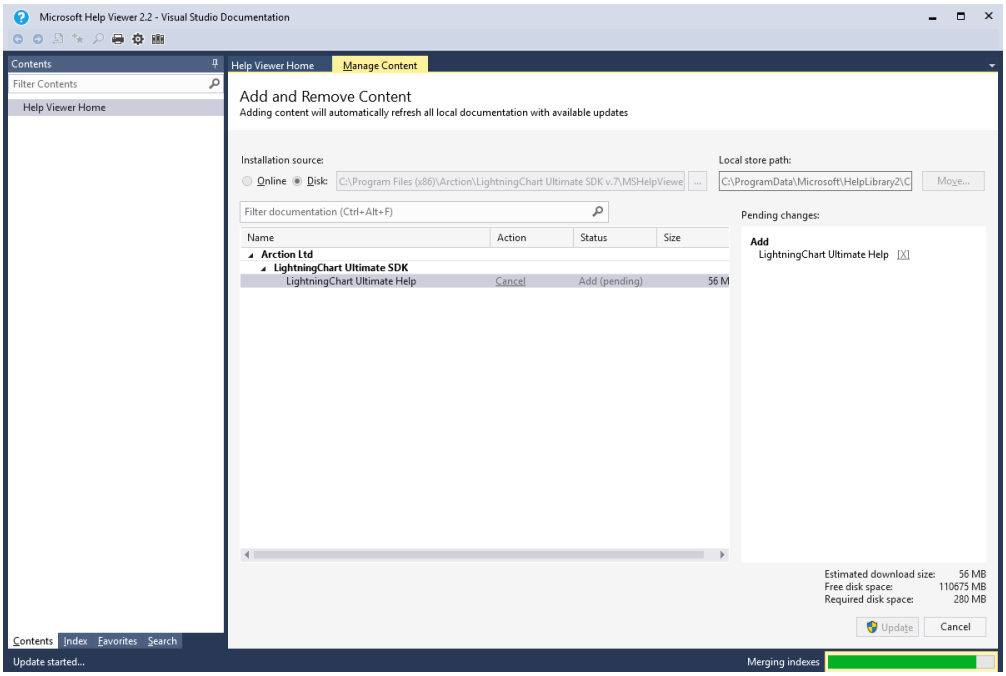


图 2-1. 添加帮助（LightningChart Help ）

9. 单击 ***Update*** 按钮。如果 Help Library Manager 询问是否要继续，单击 ***Yes*** 按钮。帮助库开始更新。
10. 帮助库更新完成后，可以关闭 Microsoft Help Viewer。
11. 在 Visual Studio Menu / Help 中, 依次选择 Set ***Help Preference : Launch in Help Viewer***。

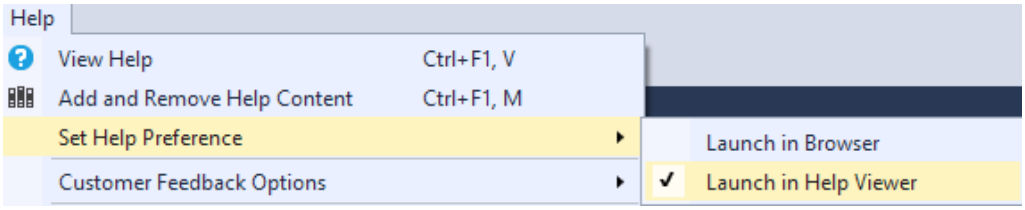


图 2-2. 设置帮助首选项（Set Help Preference）。

2.6 Visual Studio IntelliSense 的代码参数和提示

如果从 Global Assembly Cache（全局程序集缓存）引用了 LightningChart.dll 文件，但自动工具箱安装程序并未安装控件，则在键入相关 LightningChart 代码时，IntelliSense 可能不会显示代码提示。首先从项目的 References（引用）列表中移除 LightningChart.dll 文件，然后在安装目录中搜选（通常路径为 ***C:\Program files （x86）\Arction\LightningChart .NET SDK v.10\LibNet4***）再次添加。

2.7 选择 Target Framework

在 C#项目中，可以按照步骤 **Project -> Properties -> Application -> Target Framework** 来选择 framework 版本。

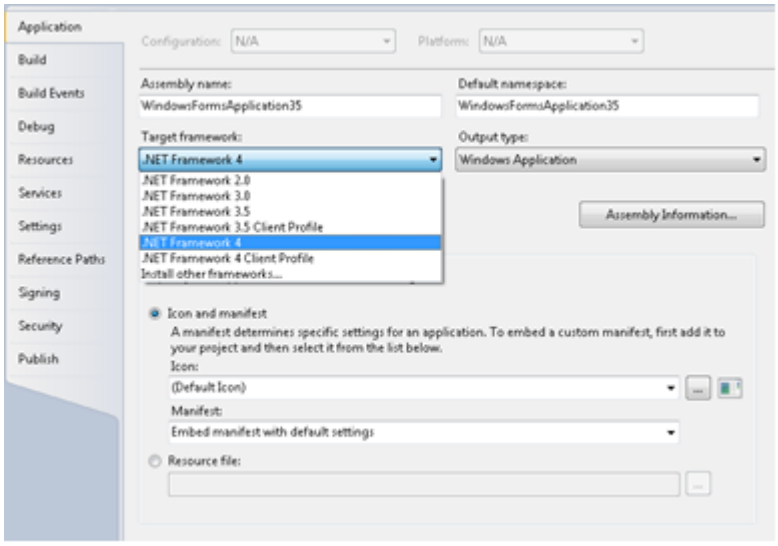


图 2-3. 在 C#项目中选择 Target Framework

在 Visual Basic 项目中，可以按照步骤 **Project -> Options -> Compile -> Advanced compile options -> Target framework** 来选择 framework 版本。

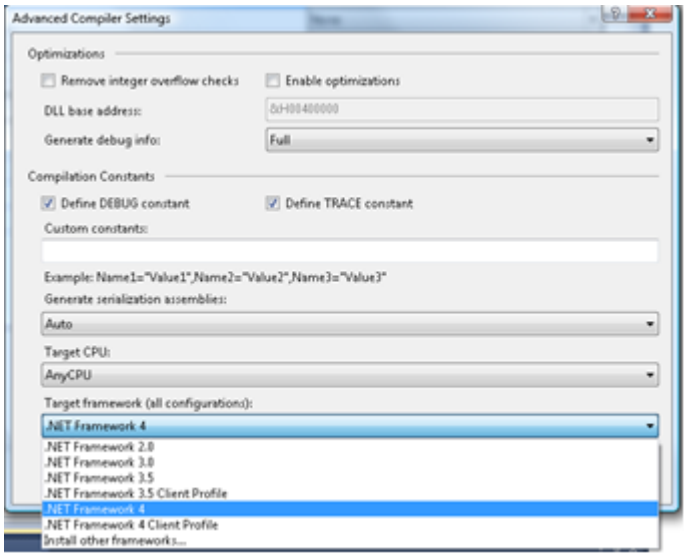


图 2-4. 在 Visual Basic 项目中选择 Target Framework

选择***.NET Framework 4 或.NET Framework 4 Client Profile***。*推荐* ***.NET Framework 4.5 或更高版本***）。

如果选择了正确的.NET Framework，Visual Studio toolbox（工具箱）中便会出现 LightningChart®.NET SDK 控件。

3. Dev Center 开发人员中心

从 LightningChart .NET v 8.5 版本开始，当运行安装程序 **LightningChart .NET SDK v10.exe** 时，Arction LightningChart .NET Dev Center 便会自动安装。Dev Center 是一款新应用程序，通过此程序可以快速访问 LightningChart®.NET 的众多功能和资源。只需单击几下鼠标键，以下任务便可轻松完成。

- 打开 ***Interactive Examples*** 应用演示。
- 创建种子项目。
- 打开说明文档（Documentation）资源，例如教程指南和使用手册。
- 通过电子邮件联系支持人员。
- 自动收集应用程序信息，发送给技术支持部门，通常有助于技术支持团队更快地解决问题。
- 快速链接，向制造商发送反馈信息。
- 检查许可证状态，打开 License Manager（许可证管理器）可对许可进行更新或激活。
- 购买新许可证。

图 3-1. Dev Center 主界面，包含了所有可能操作的按钮

3.1 打开 Interactive Examples

在学习如何使用 LightningChart 组件及其功能过程中，LightningChart 的 ***Interactive Examples*** 是学习资料的主要来源之一。点击 Dev Center 中的“Open Interactive Examples”（打开互动实例）图表，或者通过“Start”（开始菜单）的快捷方式，可以运行 ***Interactive Examples***。***Examples*** 中有按照各种不同类别进行分组的大量实例演示，并且支持通过搜索栏来搜寻具体实例。此外，每个示例具有 **Properties**（属性）标签，可以随即修改图表属性。

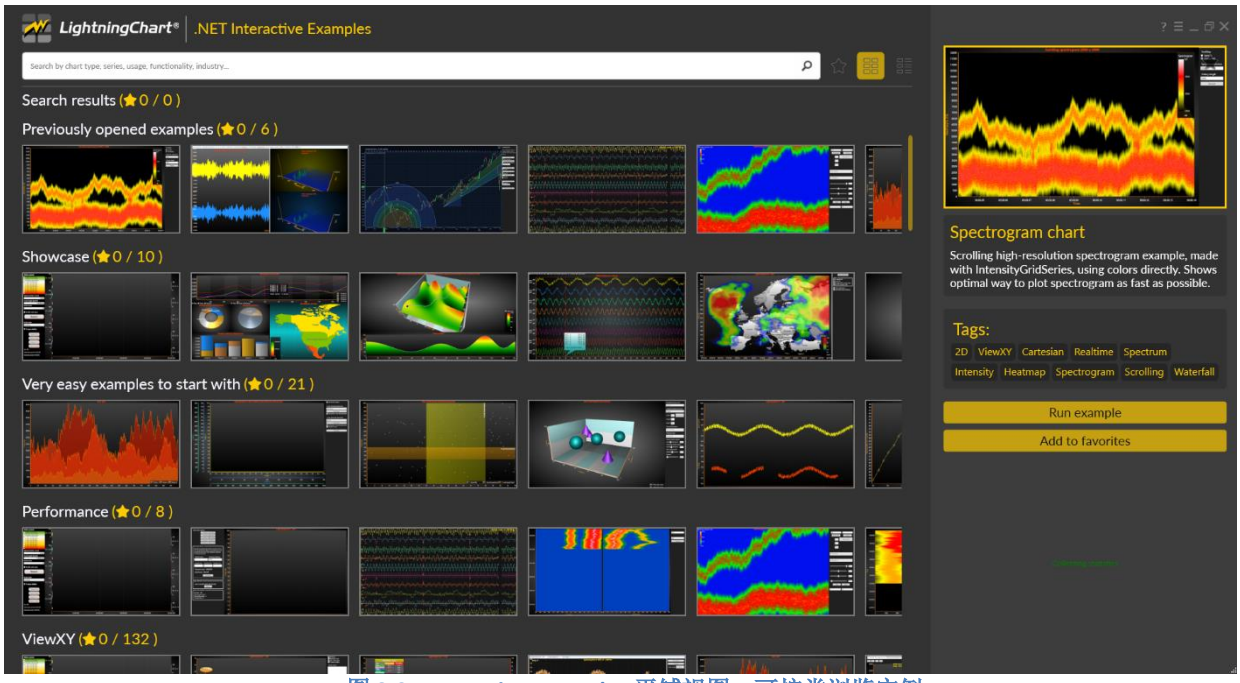


图 3-2. Interactive Examples 平铺视图，可按类浏览实例。

在安装 LightningChart .NET SDK 时，会自动将所有演示示例的源代码添加到计算机中。在 Interactive Examples 中点击“Open example VS project”（打开实例，对比项目），可以在 Visual Studio 中将当前实例作为独立项目打开并修改。

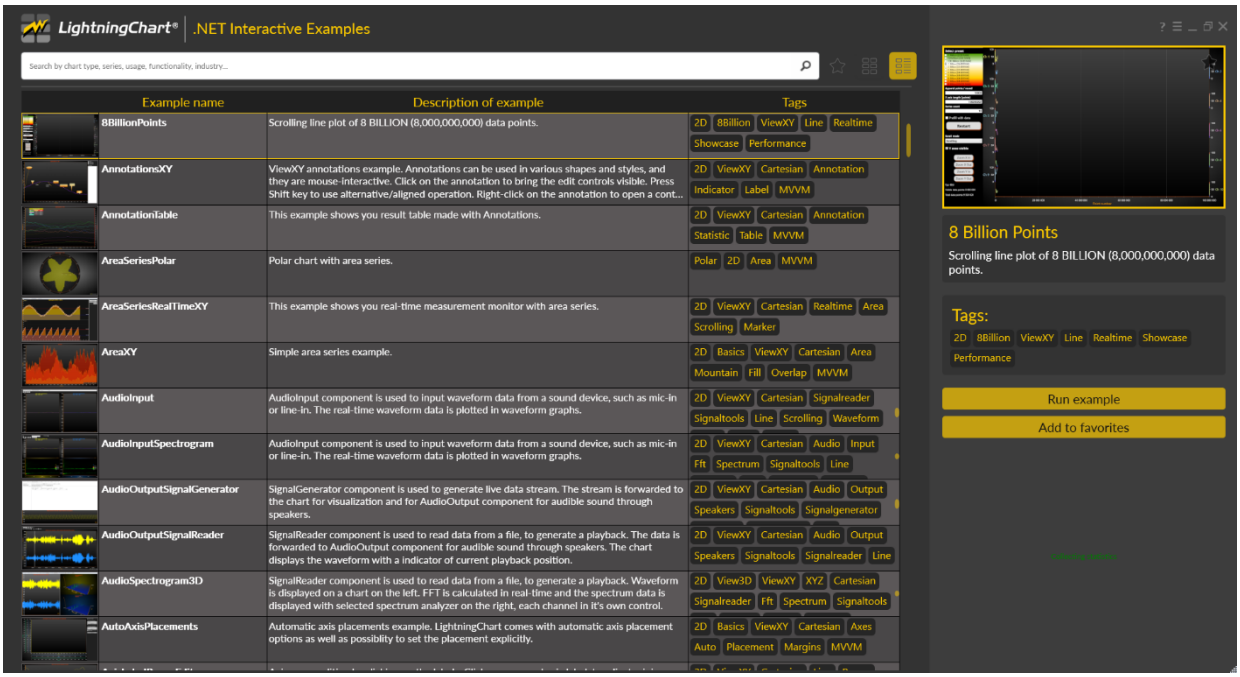


图 3-3. 列表视图，无分类展示所有实例。

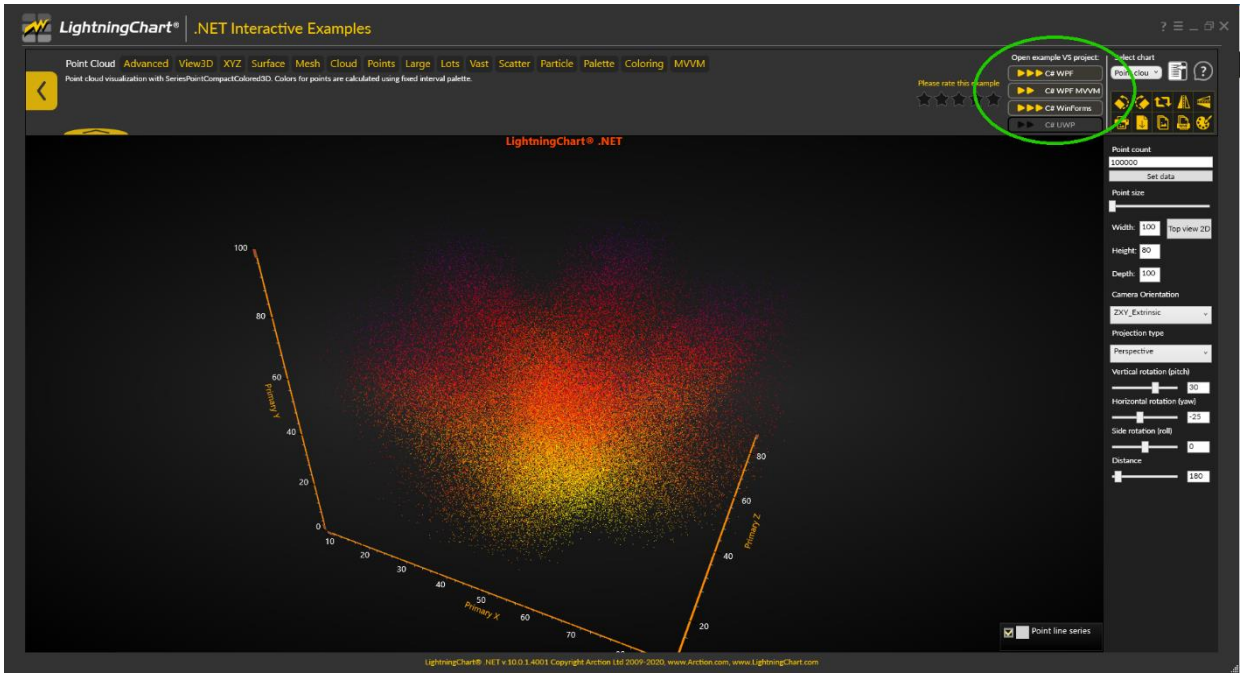


图 3-4. 图为打开的一个实例。通过突出显示区域内的按钮可以将其提取为一个独立项目。

3.2 用户数据统计

Arction Ltd.从 DevCenter 和 Interactive Examples 收集匿名用户统计信息，以改进我们的应用程序并在将来提供最佳的用户体验。 这些应用程序首次运行时，会显示用户协议。 无论您选择什么样的答案，都可以随时通过 DevCenter 或 Interactive Examples 更改此选项，并且不会影响应用程序或图表。 我们不会收集使用 LightningChart 本身的用户统计信息。

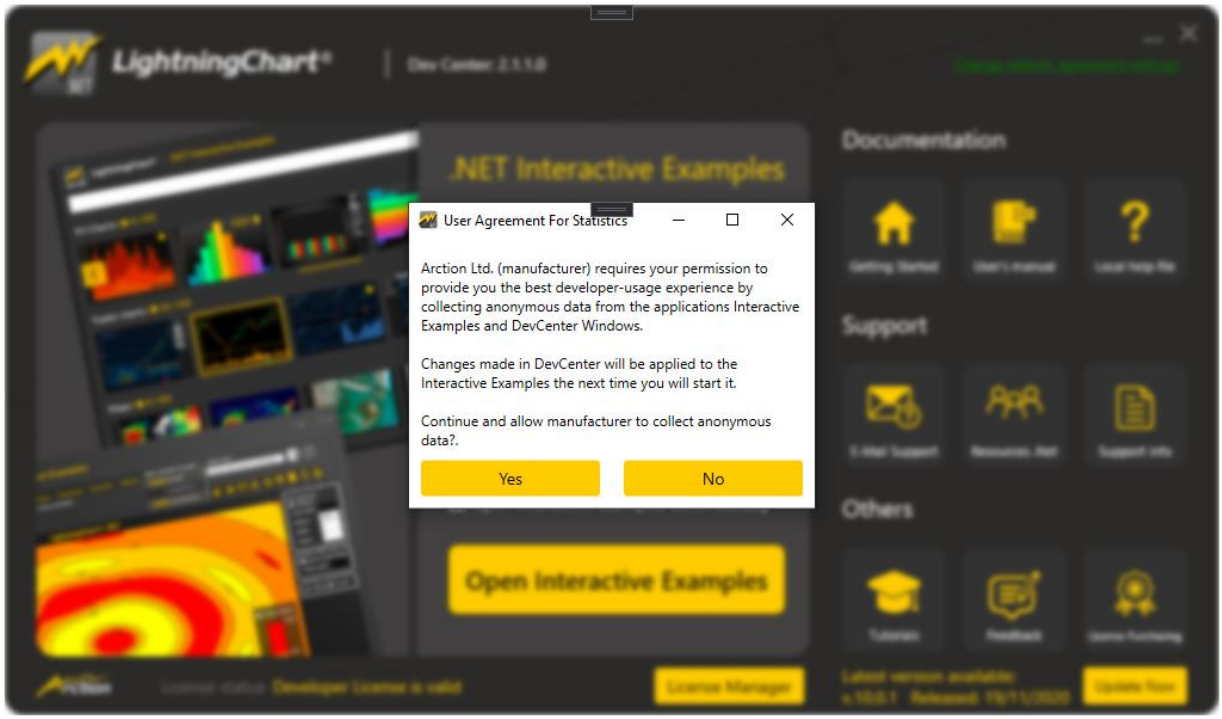


图 3-3. DevCenter 请求用户协议以收集匿名统计信息

4. 许可证管理

4.1 添加许可证

从 Dev Center 或通过 Windows 开始菜单运行 License Manager 应用程序（所有程序/ Arction / LightningChart .NET SDK v.11 / **License Manager**），可以对许可证进行管理。

Arction 组件采用了一种许可证密钥保护系统。只有通过安装有效的许可证才可以使用这些组件。许可证中包含的信息有：

- 激活各项功能，例如 ViewXY、View3D、ViewPie3D、Maps、ViewPolar、ViewSmith、Volume Rendering、Signal Tools
- WPF / WinForms / UWP / 所有平台
- 许可证可以激活到多少用户（标准为 1）.
- 订购有效日期（更新及支持服务结束日期）
- 技术支持包容性
- 单个开发者许可证或浮动型许可证
- 学生许可证

第一次将 Arction 组件从“Toolbox”拖到某个应用程序中时，在许可证管理器界面里会要求提供许可证密钥。通过单击 **Install license from file…**并浏览.alf 文件，从收到的许可证文件中添加许可证密钥。

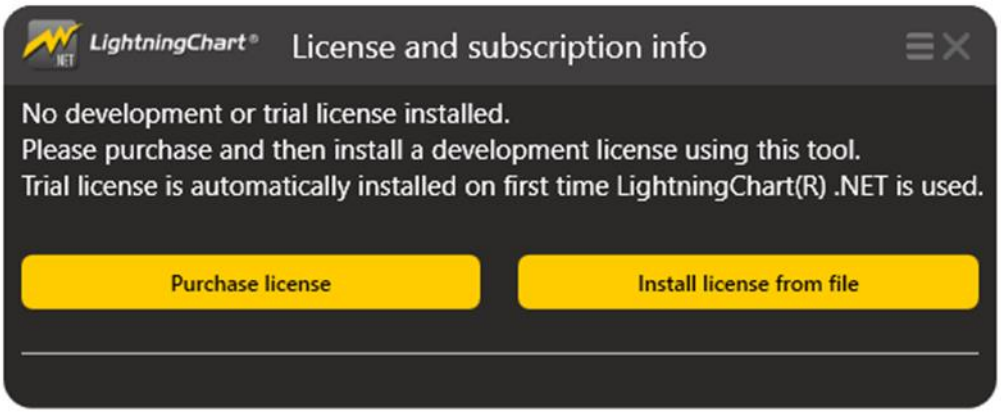


图 4-1. 没有安装许可时的 License Manager 界面。可点击 *Install license from file* 以添加许可文件。

在添加许可文件后，单个开发者许可证会自动通过 Internet 在 Arction License Server（Arction 许可证服务器）激活。

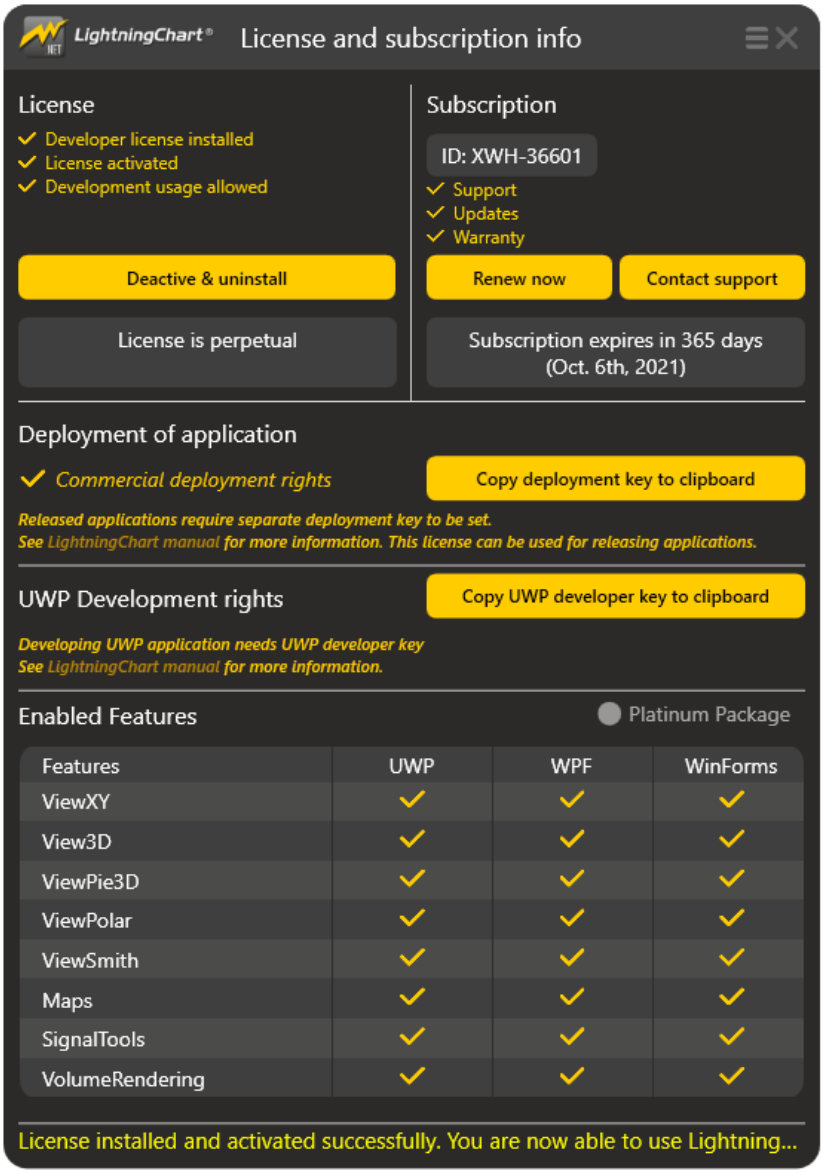


图 4-2. 成功添加一份许可文件后的 License Manager 界面

如果由于无法连接网络或者网速过慢等原因导致无法在线激活，还可以通过电子邮件激活许可证。通过网络激活的各种方法都失败后，还可以点击 **Request offline activation** 按钮来离线激活。离线停用许可证的操作原理与之类似。

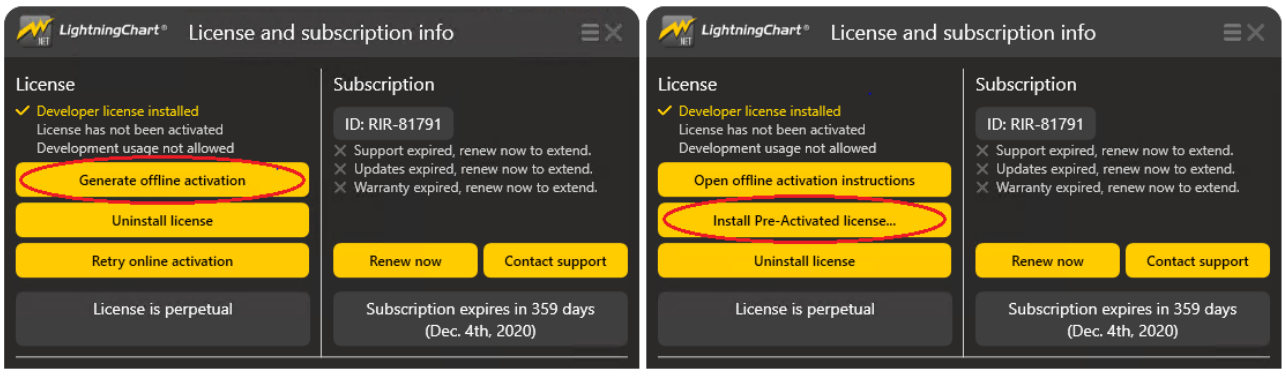


图 4-3. 在线激活失败后，可在 License Manager 中选择离线激活选项

单击离线激活按钮，屏幕上会显示操作指示。根据指示发送电子邮件至 Arction 的许可证团队：licensing@lightningchart.com.

Arction 会提供离线安装许可证的方法指导。预计 2 个工作日可收到回复。

注意! 由于密钥代码包含大量的字符，所以许可证无法通过电话激活/停用。

注意! 从 LightningChart v.7.1 版本开始，ChartManager 组件不再需要许可证密钥。

注意! 从 LightningChart v.8.0 版本开始，不再支持 LIC 格式的许可证密钥，而是需要 ALF 格式的许可证。如果您没有收到 ALF 许可证，请与 Arction 联系。

4.2 删除许可证

点击“**Deactivate & uninstall**”按钮可以将许可证从系统中移除。自动停用许可证需要有网络连接。如果无法连接到网络，可以按照之前章节的指示，通过电子邮件来停用。

许可证停用之后，可以安装到其他计算机上。

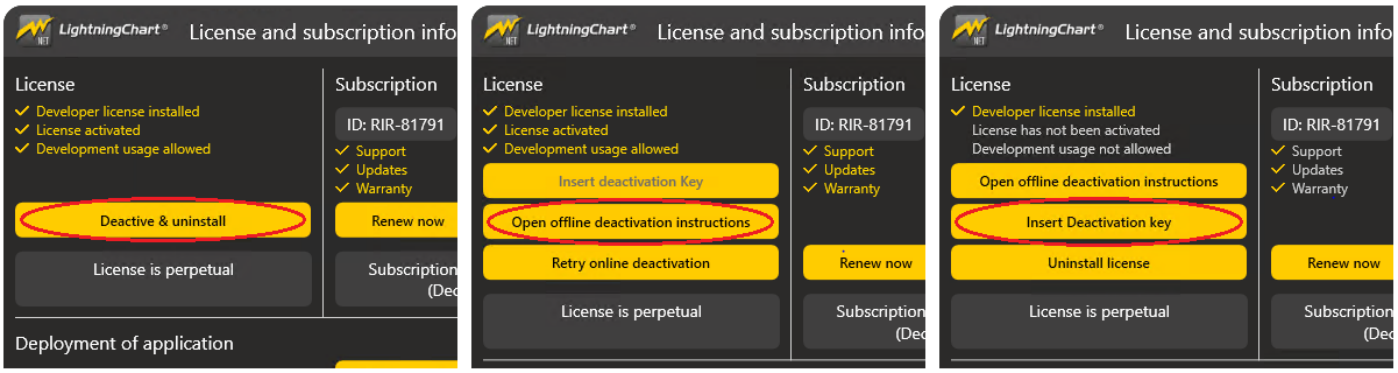


图 4-4. 停用与移除许可证。如果在线停用（左侧）失败，可以选择离线（右侧）方法

4.3 更新许可证

安装许可证后，仍然可以更新许可证，例如，延长订购期限，升级到更好的版本或购买源代码等。注意，许可证不会在用户计算机上自动更新。 因此，每个用户都应采取措施以确保开发人员计算机上的许可证是最新的。 为此，必须先停用并删除旧许可证（请参见上一章“删除许可证”如何执行此操作）。 然后，从 Arction 的客户门户获取新的许可证密钥（.alf 文件）。 然后根据第 4.1 章“添加许可证”中的说明进行安装。

4.4 提取部署密钥

要在配置了该软件的计算机中运行 LightningChart 应用程序，必须以代码形式应用部署密钥。部署密钥可以通过点击“***Copy deployment key to Clipboard***”按钮从许可证密钥中提取。

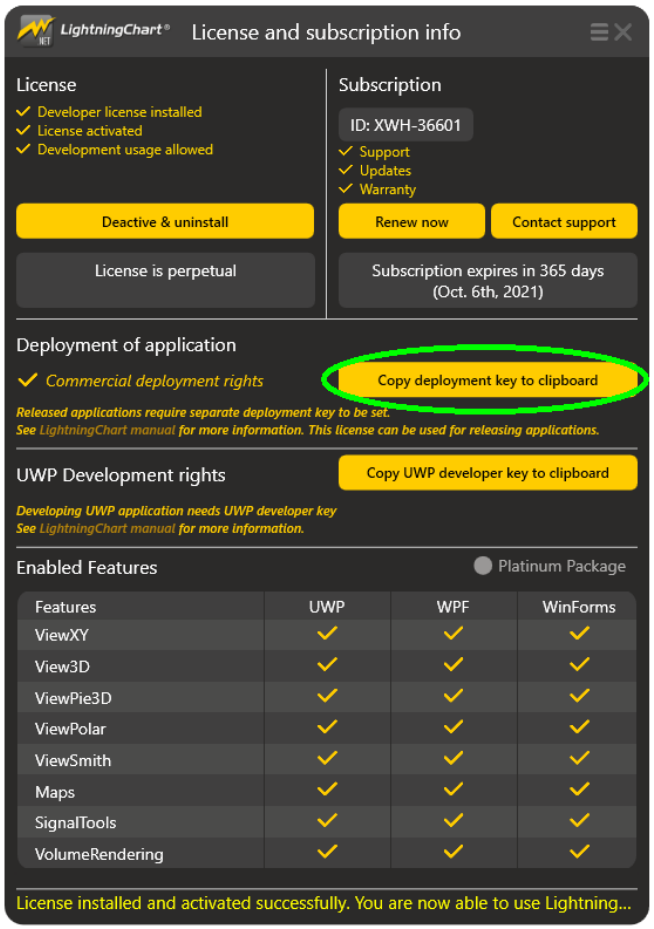


图 4-5. 将部署密钥复制到 License Manager 的剪贴板

4.5 在应用程序中应用部署密钥

在代码中，使用需要的组件时需要用到静态 ***SetDeploymentKey*** 方法。用不到的组件则不需要设置密钥（例如，在无绑定应用程序中为全绑定图表设置密钥）。在需要使用组件之前，在某处调用 ***SetDeploymentKey*** 方法。调用 ***SetDeploymentKey*** 的最佳位置是使用图表的类的静态构造函数，或者在应用程序的主类中。

更多有关发布的详细介绍，请参阅第 309 章

WinForms

以下示例展示了如何在 Program 类的静态构造函数方法（默认为每个 WinForms 应用程序创建的）上应用密钥。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
static class Program
{
static Program()
```

```
{
//Set Deployment Key for LightningChart components
string deploymentKey = "VMalgCAAO6kO1RgiNIBJABVcG.R..Kikfd...";
Arction.WinForms.Charting.LightningChart.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);
Arction.WinForms.SignalProcessing.FilterRoutines.SetDeploymentKey(deploymentKey);
Arction.CustomControls.Trader.WinForms.TradingChart.SetDeploymentKey(deploymentKey);
}
// Rest of the class ...
}
}
```

WPF

以下示例展示了如何在 *App* 类的静态构造函数中，在 App.xaml.cs 开头应用密钥。

```
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 LightningChart components
string deploymentKey = "- DEPLOYMENT KEY FROM LICENSE MANAGER
GOES HERE-";
// Setting Deployment Key for bindable chart
Arction.Wpf.ChartingMVVM.LightningChart
.SetDeploymentKey(deploymentKey);
// Setting Deployment Key for non-bindable chart
Arction.Wpf.Charting.LightningChart
.SetDeploymentKey(deploymentKey);
// Setting of deployment key to other LightningChart components
SignalGenerator.SetDeploymentKey(deploymentKey);
AudioInput.SetDeploymentKey(deploymentKey);
AudioOutput.SetDeploymentKey(deploymentKey);
SpectrumCalculator.SetDeploymentKey(deploymentKey);
```

```
SignalReader.SetDeploymentKey(deploymentKey);
FilterRoutines.SetDeploymentKey(deploymentKey);
// Setting Deployment Key for trading chart
Arction.CustomControls.Trader.WPF.TradingChart
.SetDeploymentKey(deploymentKey);
}
}
}
```

在 UWP 应用中，可以使用开发者密钥或部署密钥，但不能同时使用。在开发和调试 app 时使用开发者密钥，在部署 app 时使用部署密钥。

注意! 如果未能在应用程序中设置部署密钥，目标计算机中的 LightningChart 应用程序将会转成为期 30 天的试用模式（适用于尚未安装开发许可密钥的计算机）。

4.6 在开发计算机上使用部署密钥运行应用程序

如果在一台已经安装了开发许可证的计算机上运行应用程序（该应用程序的部署密钥已经应用了 ***SetDeploymentKey*** 方法），程序库将**优先处理开发许可证密钥**。如果部署密钥具有（包含）比本地安装的许可证（例如 Silver pack）更高级别的功能（例如 Gold pack），可能会对用户造成困扰或扰乱调试。开发人员必须注意这一限制因素。

Arction 建议整个团队中的所有许可证都是同一类型。

4.7 在有调试程序的情况下运行应用程序

正确设置了部署密钥后，当从带有调试程序的 Visual Studio 运行项目时，如果系统中没有发现开发许可证，那么图表便会进入慢渲染模式，最大 FPS 为 1，并且在图表上方会显示消息文本。

LightningChart EULA 禁止使用不具备开发人员许可证密钥的 LightningChart 直接进行开发或调试。

4.8 试用期

试用期为 30 天。 试用期结束后，必须购买许可证才能继续使用该产品。待更新了正确的许可证后，在试用许可证期间建立的所有项目还可以继续使用。当运行使用试用许可证建立的图表应用程序时，会显示有试用版本才会出现的消息不断提醒用户付费使用。

4.9 浮动许可证 （Floating licenses）

浮动许可证可以安装的计算机数量没有限制。Arction 已对并行开发的人员数量进行了配置。只有付费的并发用户，才能同时使用 LightningChart 进行开发。当一位开发人员利用 LightningChart 完成开发工作之后，在另一位开发人员可以开始使用之前，大约有 10-15 分钟的暂停时间。

设置部署密钥必须与设置单个开发者许可证类似。

默认情况下，浮动许可证由 Arction Licensing Server 管控，在进行开发时，需要持续连接网络。

另外还可以购买客户端浮动许可证控制器。未付费前，开发计算机是通过局域网连接到在客户单位中运行的服务，无法与 Arction 或其他的部分进行在线通信。购买许可证后，Arction 会另行提供有关安装控制器服务和浮动许可证的使用说明。

5. LightningChart 组件

5.1 使用 LightningChart®.NET 函数库

为使用 ***LightningChart®.NET*** 组件，必须添加引用 Arction .dll 文件。这些.dll 文件可在安装文件夹中找到。在开发应用程序时需要用到以下程序集。

Winforms:	LightningChart.WinForms.Charting.NET4.dll.
WPF 无绑定:	LightningChart.Wpf.Charting.NET4.dll LightningChart.DirectX.dll
WPF 绑定:	LightningChart.Wpf.ChartingMVVM.NET4.dll LightningChart.DirectX.dll
UWP Chart:	LightningChart.Uwp.ChartingMVVM.dll
如果使用 SignalTools:	LightningChart.WinForms.SignalTools.NET4.dll or LightningChart.Wpf.SignalTools.NET4.dll or LightningChart.UWP.SignalTools.dll

使用 .NET6 程序集时，NET4 更改为 NET6。 否则，名称相同

以上参考文件添加成功后，在建立项目时，全部所需的程序集会自动复制到输出文件夹中。第 28 章介绍了在开发一个 LightningChart 应用程序时需要用到哪些程序集。

由于 Arction.DirectXFiles.dll 文件过大，会增加初始化时间，添加引用文件时不会自动包括在内。只有当系统中没有正确的 DirectX 程序集的时候才需要添加。Arction.DirectXInit.dll 程序会检查已有的.dll 文件，并在需要时加载这些文件。一经加载后，会将 DirectX-dlls 文件写入到 Windows 的缓存文件夹，供 LightningChart 在将来需要时访问使用，这样就加快了初始化速度。

建议不要把 Arction.DirectXFiles.dll 列入引用文件，而是将其复制到 exe.文件。

5.2 用代码创建图表

LightningChart 组件可以通过从 toolbox 中拖拽来添加，或者完全采用代码隐藏（代码后置）方式来添加。以代码形式来创建图表对象具有更方便地更新版本的优势。此外，还可以避免一些与（反）序列化相关的问题。

以下示例演示了一种以代码隐藏方式（.xaml.cs 文件）创建一个 WPF 无绑定图表的方法。

```
using LightningChartLib.WPF.Charting;

namespace ExampleProject
{
    public partial class ExampleApp : Page
    {
        private LightningChart _chart = null;

        public ExampleApp()
        {
            InitializeComponent();

            CreateChart();
        }

        private void CreateChart()
        {
            _chart = new LightningChart();

            //父容器中的图表控件
            (Content as Grid).Children.Add(_chart);

            //图表未正确设置，禁用渲染。
            _chart.BeginUpdate();

            // 配置图表。

            // 渲染图表。
            _chart.EndUpdate();
        }
    }
}
```

5.3 从工具箱添加至 Windows Forms 项目

将 ***LightningChart*** 控件从 toolbox 添加至设计窗中，会出现一个图表，其属性显示在 ***Properties*** 窗口中。

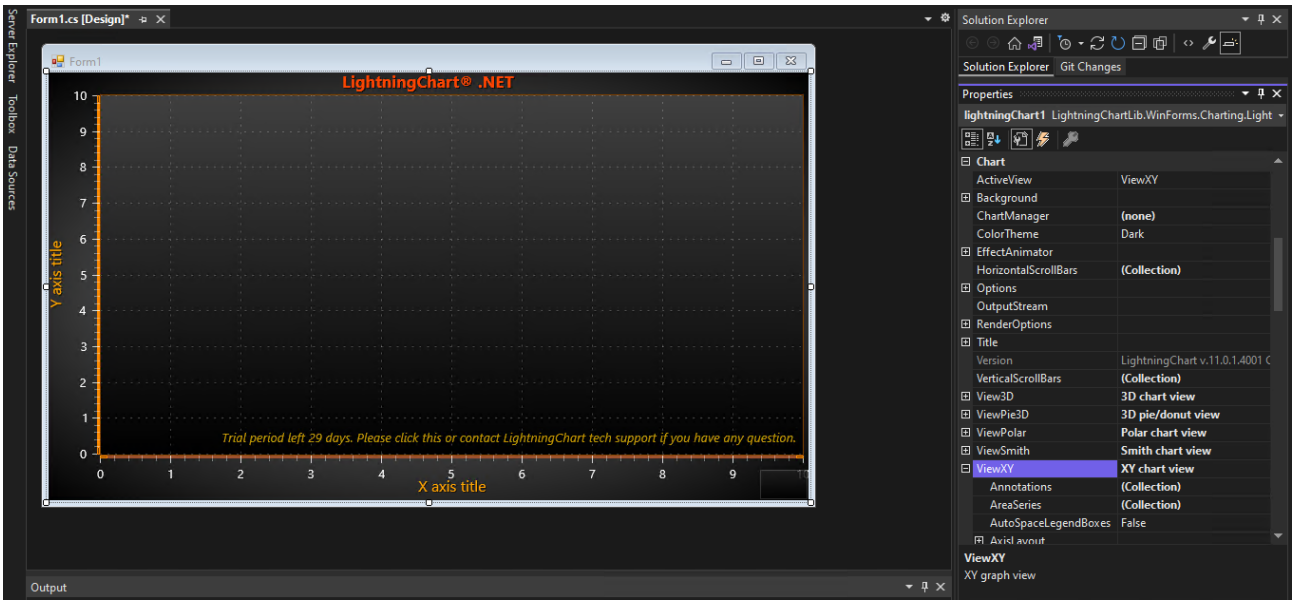


图 5-1. 在 Windows Forms 设计窗中添加 LightningChart 控件

5.3.1 属性

对属性可以进行自由修改。此外，还可以将新的系列和其他对象插入到其集合中。系列数据点必须通过代码表示。

5.3.2 事件处理程序

图表主级别的事件处理程序，可以使用属性网格进行分配。对于已添加至集合的对象而言，必须通过代码来分配事件处理程序。

5.3.3 有关更新版本的最佳方法

图表属性数据经过序列化，存放至 Visual Studio 项目的***.resx*** 文件中。版本更新后，LightningChart API 会出现一点小变化，可能导致在***.resx*** 文件中新版本存在不兼容的序列化结果。

为了便于更新，强烈建议创建图表对象，以代码形式添加所有系列、事件处理程序等。然后项目正确加载，在编译时显示可能的错误，这让修复这些错误比修复.resx 文件更容易。采用.resc 文件，一些属性定义可能会丢失，但采用代码，这些属性的具体定义一直都不会变。

5.4 从工具箱添加至 WPF 项目

从 toolbox 向 Window 或另一个容器添加 ***LightningChart*** 全绑定 WPF 控件。图表会出现在设计窗口，并且其属性会在 ***Properties*** 窗口显示出。XAML 编辑器可以显示出具体内容以及对图表默认属性做出的修改。

5.4.1 属性

对属性不仅可以进行自由修改，而且可以将新的系列以及其他对象插入到其集合中。系列数据点必须采用代码形式表示。

5.4.2 事件处理程序

图表主级别的事件处理程序，可以使用属性网格进行分配。对于已添加至集合的对象而言，必须通过代码来分配事件处理程序。

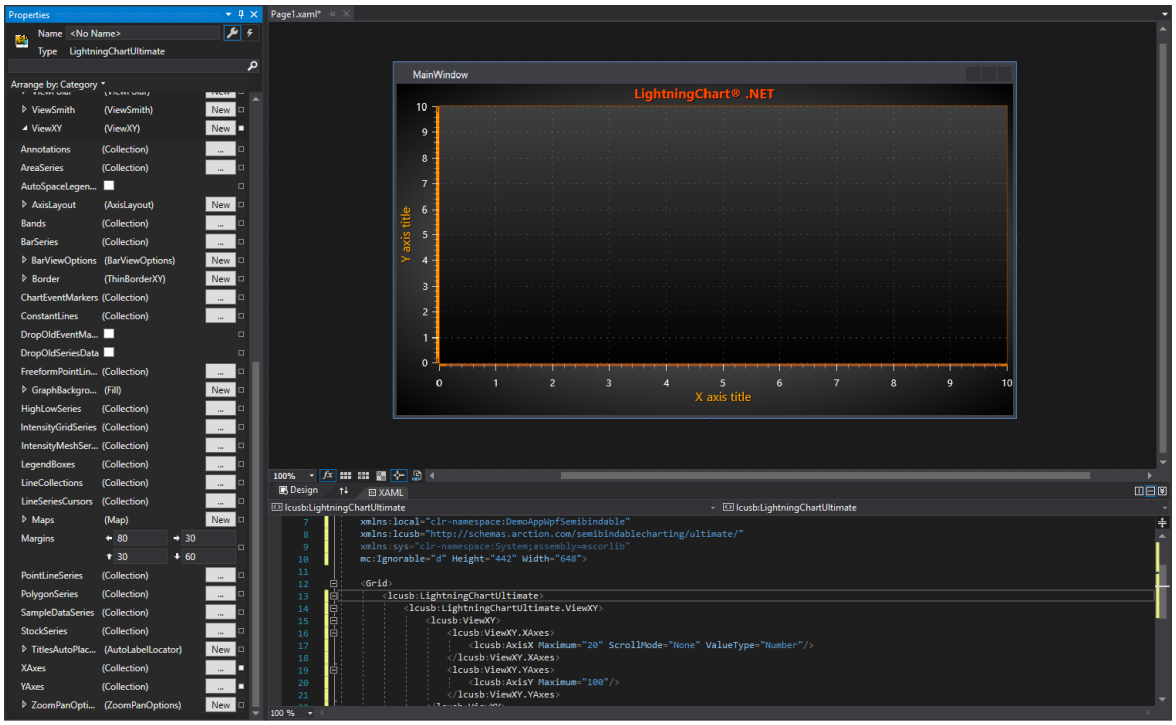


图 5-2. 添加 LightningChart 控件至 WPF 设计窗.

5.5 添加至 Blend WPF 项目

在“***Projects***”选项卡中，选择“***References***”。右键单击并选择“***Add reference…***”，然后浏览，从 ***c:\program files （x86）\Arction\LightningChart .NET SDK v.10\LibNet4 中搜寻到*** Arction.WPF.Charting.LightningChart.dll 文件。

选择“***Assets***”选项卡。然后在搜索框中输入“Lightning”。在搜索结果中会显示带有 ***LightningChart*** 的行。再将对象拖动到 WPF 窗口中。

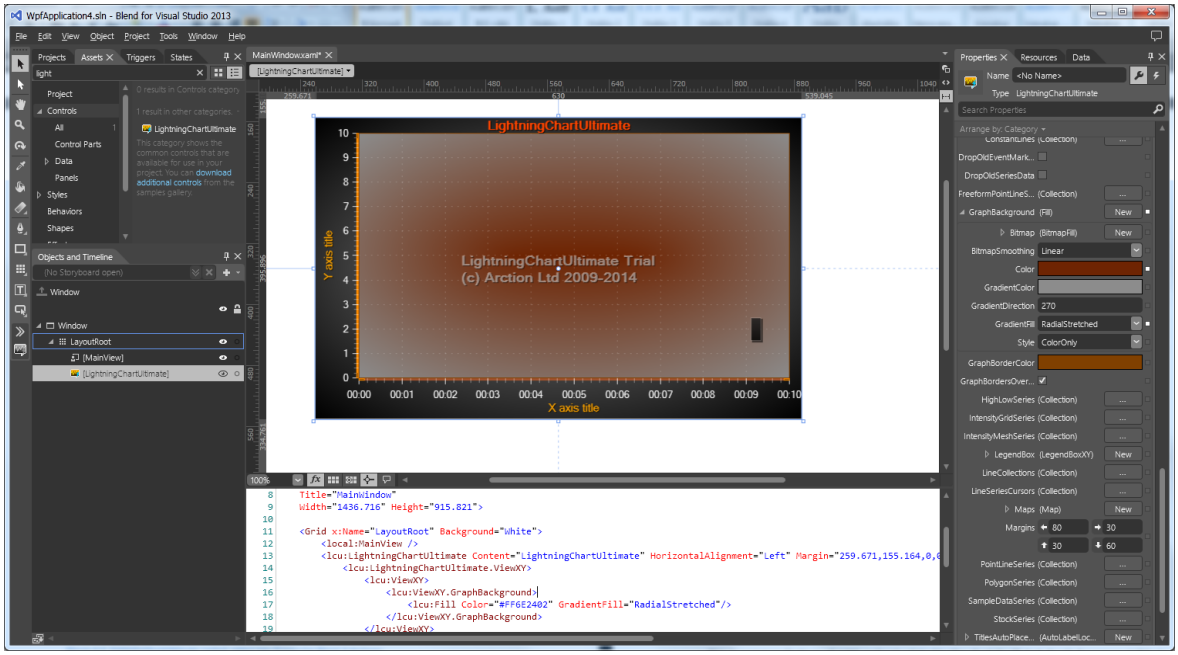


图 5-3. 将 LightningChart 控件添加至 Blend For Visual Studio 2013 设计窗中

5.5.1 有关版本更新的最佳方法

图表属性数据采用 XAML 存储。新版本的属性设置略有不同，会导致 LightningChart 对象不会出现在设计窗中。随后需要进行相关的 XAML 修改。XAML 标签树可能会非常庞大，并且可能会非常难以进行编辑。

为了便于更新，强烈建议在设计窗中创建图表对象，设置好其布局，并对齐相关属性。使用代码设置其他部分。或者图表对象也使用代码创建。

5.5.2 防止图表模糊

这是 WPF 的一个共同特征，而且与图表本身无关，但采用 LightningChart 精确渲染后会变得清晰明显。

设置图表父控件 ***UseLayoutRounding = True***，可以防止图表看上去模糊不清。虽然图表在设计窗中可能仍然看上去模糊，但是当运行应用程序后会看的很清晰。父控件例如 ***Grid***, ***Canvas***, ***DockManager*** 等。

5.6 创建 UWP 项目

在 UWP 中使用 LightningChart 与在全绑定 WPF 图表使用的工作原理类似，因为都具有类似的绑定和 MVVM 功能。UWP 图表的属性集合与全绑定 WPF 图表一样，（比如 ViewXY 轴、3D 光）默认为空值。Windows 10 和 Visual Studio 2017 或 2019 需要使用 LightningChart 来开发 UWP 应用程序。Universal Windows Platform 开发也应该安装在 Visual Studio 上进行。

- Microsoft.NETCore.UniversalWindowsPlatform：6.2.8 或更高版本（Nuget 程序包）。

- Microsoft.Toolkit.Uwp：v 4.0.0 或更高版本，建议 6.0.0 或更高版本。 请注意，最新的工具包可能与较早的目标版本不兼容。

5.6.1 创建 UWP 应用

利用 LightningChart 按照以下步骤创建 UWP 应用程序:

1. 用 Visual Studio 创建一个新项目。选择 ***Blank App (Universal Windows)***.
2. 给项目命名，并选择文件位置。
3. 为项目设置 ***Target*** 和 ***Minimum*** 版本，具体要根据电脑上安装的 SDK 来选择。了解更多信息，可参阅 Microsoft 文件: https://developer.microsoft.com/en-us/windows/downloads/sdk-archive/. 推荐 16299 版及以上版本。注意，之后通过 Project -> Properties 可进行更改。

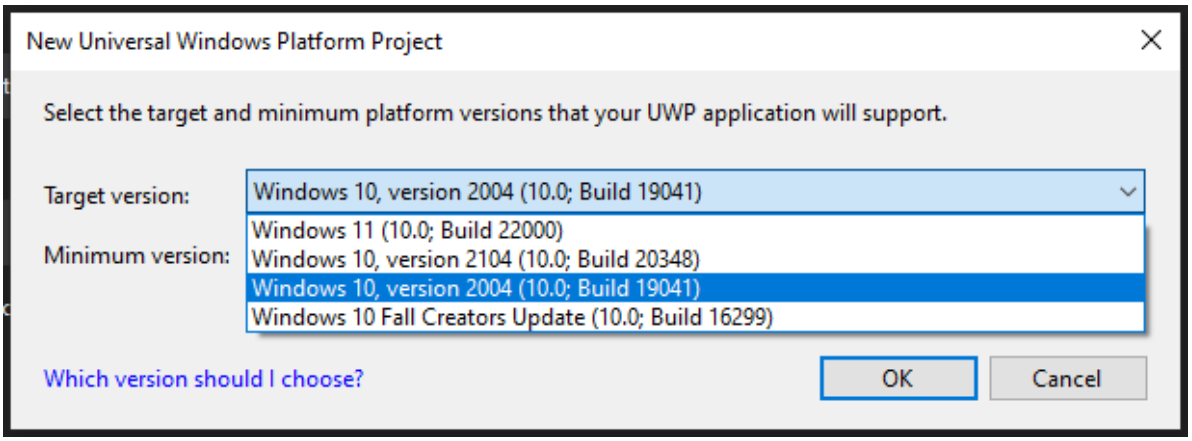


图 5-4. 为 UWP 选择 Target 和 Minimum 版本。

4. 使用 Target 版本 2004 或更新版本时，应禁用 TypeInfoReflection 设置。 打开.csproj 项目文件例如在文本编辑器中，并将以下行添加到每个 PropertyGroup 定义构建条件（Debug|x86、Release|x86 等）：
   <EnableTypeInfoReflection>false</EnableTypeInfoReflection>
5. 将 LightningChart 和 SharpDX 组件添加到参考中。 默认情况下，这些可以在 ***C:\Program Files\LightningChart\LightningChart .NET SDK v.11\LibUWP***. 到 。 注 意 同 UWPCopyright Arction Ltd 2009-2021 47 程序集适用于 x86、x64、Arm 和 Arm64 平台。 可以通过以下方式更改目标平台：右键单击项目并选择 Properties -> Build -> Platform target

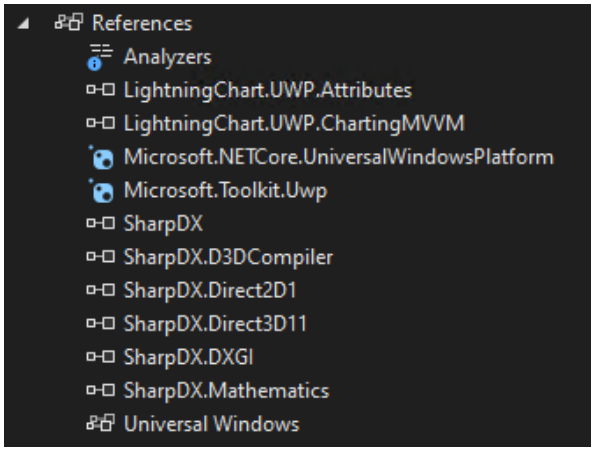


图 5-5. UWP 项目的参考。 LightningChart 和 SharpDX 文件是从 LightningChart 安装文件夹添加的。

6. 为项目安装 ***Microsoft.Toolkit.Uwp*** NuGet 程序包。推荐 6.0 版或更新版本。
7. UWP 需要在应用程序中设置开发者密钥。通过“***Copy UWP developer key to clipboard***” 按键从 LicenseManager 提取密钥，然后用 ***LightningChart.SetUwpDeveloperKey()***方法在 ***App.Xaml.cs*** 文件中进行设置。

图 5-6. 用 LicenseManager 将 UWP 开发者密钥复制到剪切板。

```
using Arction.Uwp.ChartingMVVM;

namespace ExampleProject
{
    /// <summary>
    /// Provides application-specific behavior to supplement the default Application class.
    /// </summary>
    1 reference
    sealed partial class App : Application
    {
        /// <summary>
        /// Initializes the singleton application object.  This is the first line of authored code
        /// executed, and as such is the logical equivalent of main() or WinMain().
        /// </summary>
        0 references
        public App()
        {
            LightningChart.SetUwpDeveloperKey("Copy developer key here");

            this.InitializeComponent();
            this.Suspending += OnSuspending;
        }
```

图 5-7.在 App.Xaml.cs 文件中设置 UWP 开发者密钥。

8. 现在可以在代码或 xaml 编辑器中创建 LightningChart 组件。图表。示例，用代码创建 UWP：

```
using LightningChartLib.Uwp.ChartingMVVM;

namespace ExampleProject
{
    public sealed partial class MainPage : Page
    {
        private LightningChart _chart = null;

        public MainPage()
        {
            InitializeComponent();

            CreateChart();
        }

        private void CreateChart()
        {
            _chart = new LightningChart();

            //Chart control into the parent container.
            (Content as Grid).Children.Add(_chart);

            //Disable rendering until the whole chart is set up correctly.
            _chart.BeginUpdate();

            //configure chart here.

            // Allow rendering the chart
            _chart.EndUpdate();
        }
    }
}
```

9. 构建、部署并运行应用程序。如果应用程序不可运行（例如出现“Activation of the Windows Store app”错误），往往更改 Target 和 Minimum 版本即可解决问题。
10. 在向其他电脑部署 UWP 时，须应用部署密钥（参阅第 4.4 章节）。部署密钥不能与开发密钥一起使用。所以，在部署时要删除开发者密钥设置。

5.6.2 UWP 疑难解答

UWP 项目有一些已知问题。 这些通常与 LightningChart 无关，但通常与 UWP 相关。 因此，建议从网上搜索其他信息。 在任何情况下，如果您有任何问题，请随时联系支持人员 (support@lightningchart.com)。

一些已知的 UWP 问题：

- 版本 1903 不起作用。可能会出现错误，例如构建错误 MSB4166： *Child node "2"*过早退出

这是 1903 特有的已知问题。最好的解决方法是简单地使用不同的目标版本。 Microsoft 建议改为以 2004 版（内部版本 19041）为目标。

- 发布版本不起作用。

尝试通过项目的 Properties -> Build 禁用 Compile with .NET Native 工具链

- 调试构建不起作用

在某些 UWP 版本（例如 2004 版）中，调试版本可能无法运行并给出错误：*Run Error: an unhandled win32 exceptio occurred in [3088] (number is different for different build).*。在这些情况下，请确保您已将 <EnableTypeInfoReflection>false</EnableTypeInfoReflection> 行添加到您的 csproj 文件中。或者，尝试改用发布版本。

- 激活 Windows 应用商店应用“*App name*”失败。

尝试清除方案。删除 bin 和 obj 文件夹并按照此处的说明重建：

https://docs.microsoft.com/en-us/previous-versions/hh972445(v=vs.140)?redirectedfrom=MSDN

5.7 XAML 序列化兼容性 5.7 XAML 序列化兼容性

从版本 12.0 开始，LightningChart .NET 支持 XAML 序列化，这意味着 .NET 类 XamlWriter 可用于将 LightningChart 对象序列化为 XAML 标记。此外，.NET 类 XamlReader 可用于读取 XAML 输入并基于它创建 LightningChat 对象。通过使用 XamlWriter.Save() 和 XamlReader.Load() 方法，用户可以保存图表

的状态（保存大多数属性的值），并在下次打开应用程序时导入该状态。请注意，系列数据或点未序列化。此外，VolumeModel 属性未序列化。

XAML 序列化允许使用非可绑定版本的 LightningChart 在 *.xaml 代码中创建图表，但如果需要绑定图表属性，则应使用 MVVM 版本的
LightningChart

图 58. 序列化图表的 XAML 字符串

5.8 对象模型

学习 LightningChart 对象模型的最佳方法是使用交互式示例应用程序/演示的属性编辑器。 只需打开演示，选择示例并打开所选图表的属性编辑器

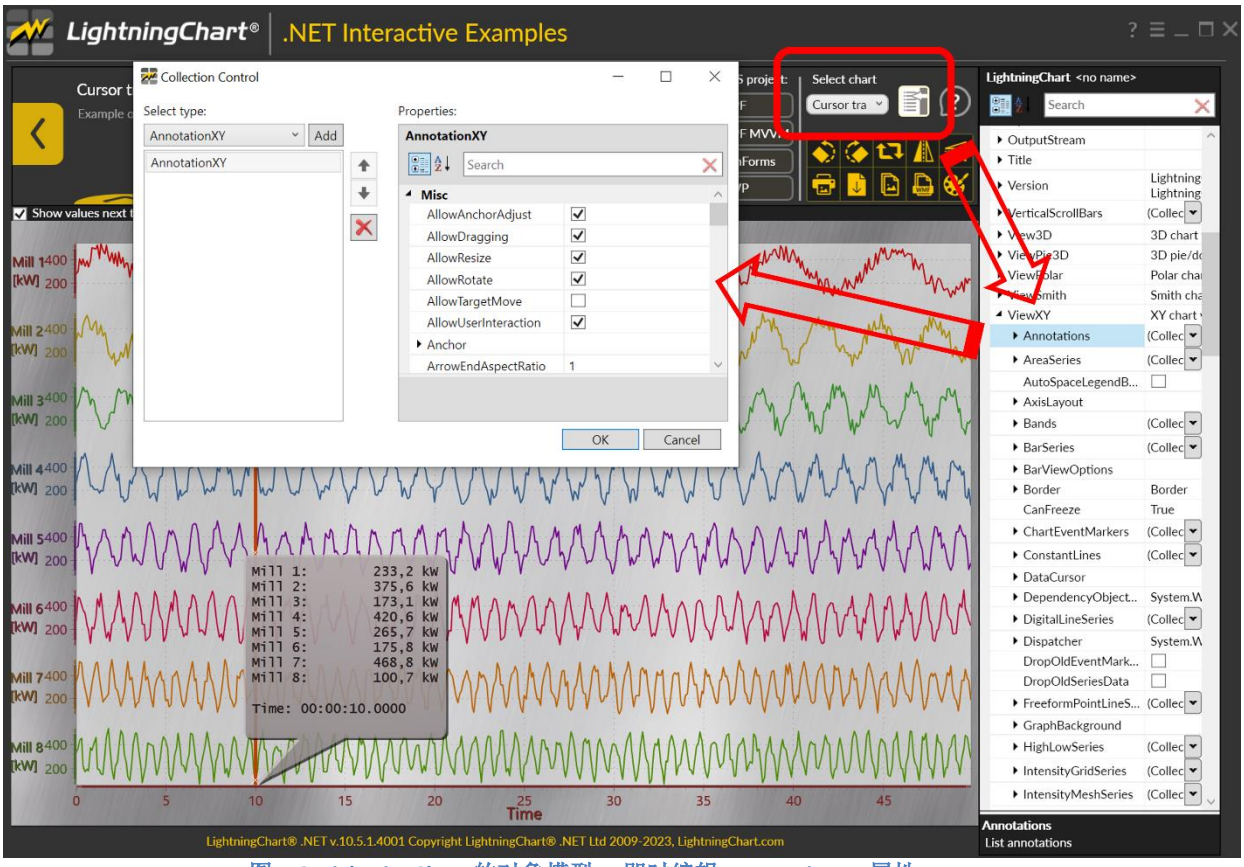


图 5-8. LightningChart 的对象模型。 即时编辑 AnnotationXY 属性

在软件开发过程中，在应用程序运行时访问属性会很方便。 用户可以在自己的独立应用程序中创建类似于上述 LightningChart 演示的属性编辑器。

对于 WinForms 应用程序，在源代码中添加以下方法调用：

```
chart.ShowPropertiesEditor()
```

对于 WPF 应用程序，可使用 Extended WPF Toolkit™创建属性网格。

5.8.1 Windows Forms、 WPF 及 UWP 之间的区别

就 Chart 一级而言，Windows Forms 和 WPF 的属性树以及对象模型几乎是完全相同的。二者主要不同之处如下；

Windows Forms	WPF	UWP

Rendering options property	RenderOptions	ChartRenderOptions	ChartRenderOptions
Background fill property	Background	ChartBackground	ChartBackground
Fonts	System.Drawing.Font	LightningChart.WPF.Charting.WpfFont	LightningChart.UWP.ChartingMVVM.UwpFont
Colors	System.Drawing.Color	System.Windows.Media.Color	Windows.UI.Color

表 4-1. Windows forms、 WPF 与 UWP 的不同之处

在接下来的章节中，除非另有说明，否则将引用 Windows Forms 的属性名称。

5.9 LightningChart 视图

LightningChart 的主要视图如下：

- ViewXY （参阅第 6 章）

- View3D （参阅第 7 章）

- ViewPie3D （参阅第 9 章）

- ViewPolar （参阅第 10 章）

- ViewSmith （参阅第 11 章）

通过设置 ***ActiveView*** 属性可以对可视化视图进行更改。默认视图为 ViewXY。

```
// 设置可视化视图为3D
chart.ActiveView = ActiveView.View3D;
```

5.10 视图与缩放区域定义

LightningChart 视图包含了几个不同的区域，各自反映了不同的信息。这些区域根据视图的内容可以看做是几个平面矩形，但无论视图类型如何，这些定义都是一致的；特别是在缩放操作时，将决定要显示图表的哪部分区域。

-ChartArea/ViewArea: 整个区域包含图表及图边距。

-MarginRectangle: MarginRectangle （或 MarginRect）包括图边距内部的区域.

-GraphArea: 该区域由轴范围界定。包含主网格及次网格。除非某些数据值超出了轴范围，否则数据会绘制在这个区域中。

-Background area / circle: 基本上和 GraphArea 一样。还包含轴范围与网格之外的图形部分。

-LabelsArea: 该区域包括图形及轴标签。忽略了数据。

-Data: 该区域仅包含数据。由数据的最大与最小值决定。

-DataAndLabelsArea: 结合了 Data 和 LabelsArea 两个部分。包括了所有的数据、轴、标签以及标记。

-Border: 一个可自定义的 1 像素宽的矩形，指示出图边距的位置。通过禁用/启用可以显示或者隐藏。

-Margins: 图边距是图形区域周围的空白区域。视图的大部分内容都填置在图边距内，图边距外部分都会被剪切掉。

-ZoomPadding: 该区域是在缩放操作后，图边距与另外的预定义区域之间留出的空间（请参阅第 7.19.3 章节）。在 ViewXY 视图中没有作用。

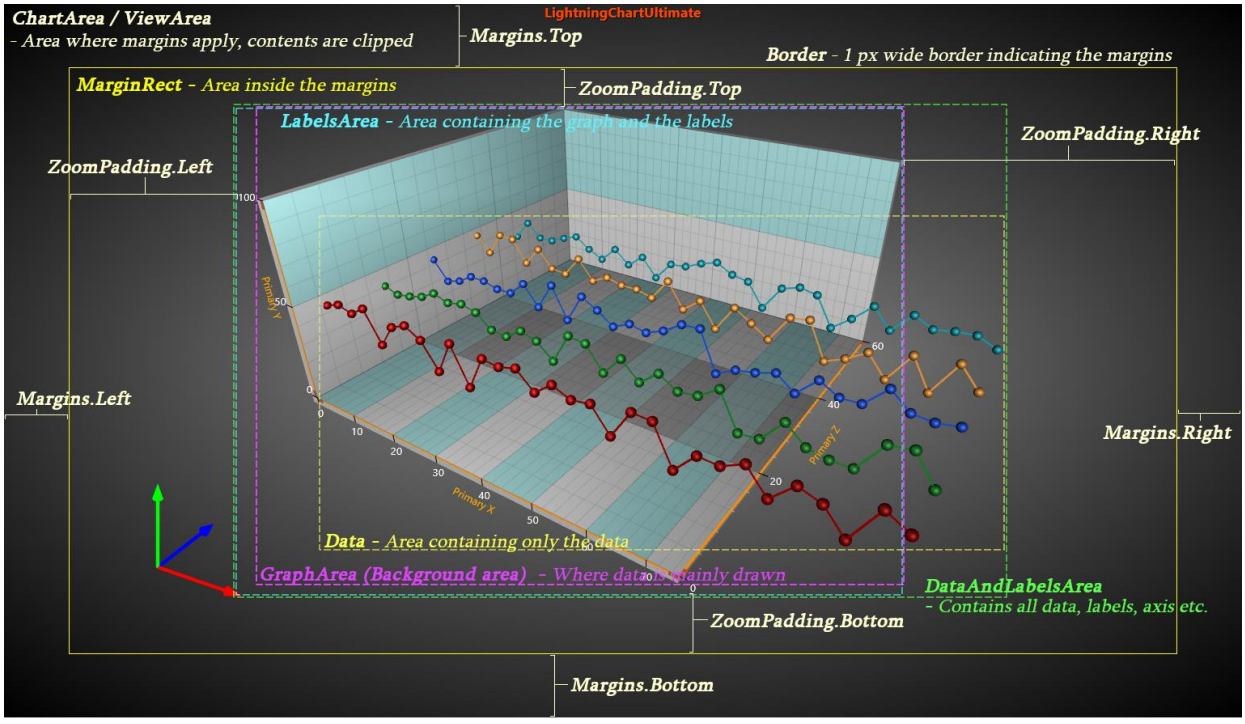


图 5-9. 视图与缩放区域的定义

5.11 设置背景填充

所有的视图的背景填充都一样。

- 在 WinForms 中使用 ***chart.Background***

  chart.Background.Color = Color.DarkBlue;

- 在 WPF 中使用 ***chart.ChartBackground***

  chart.ChartBackground.Color = Colors.DarkBlue;

背景填充支持以下几种方式：

- 纯色填充。设置 ***GradientFill = Solid***，然后用 ***Color*** 来定义颜色。
- 渐变填充，改 ***Color*** 为 ***GradientColor。设置*** GradientFill = Linear / Radial / RadialStretched / Cylindrical，用 ***GradientDirection*** 来控制渐变方向为 Linear（线性）或是 Cylindrical（圆柱形）。

  chart.ChartBackground.GradientFill = GradientFill.Cylindrical;
  chart.ChartBackground.GradientColor = Colors.Black;
  chart.ChartBackground.GradientDirection = -45;

- 位图填充，有几个不同的平铺和拉伸选项。还支持位图着色与透明度调节，可设置半透明填充位图。

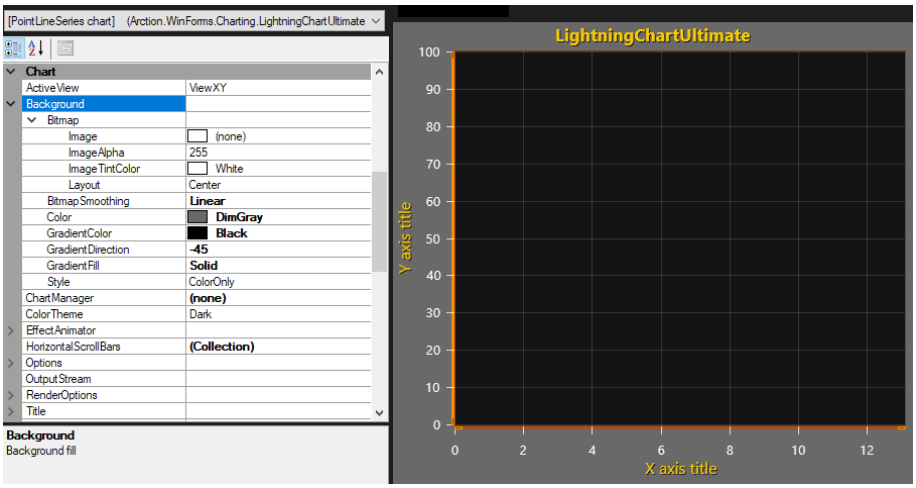


图 5-10. 在 Background 中为背景选项，在 ViewXY 列下为参数。 GradientFill = Solid， Color = DimGray

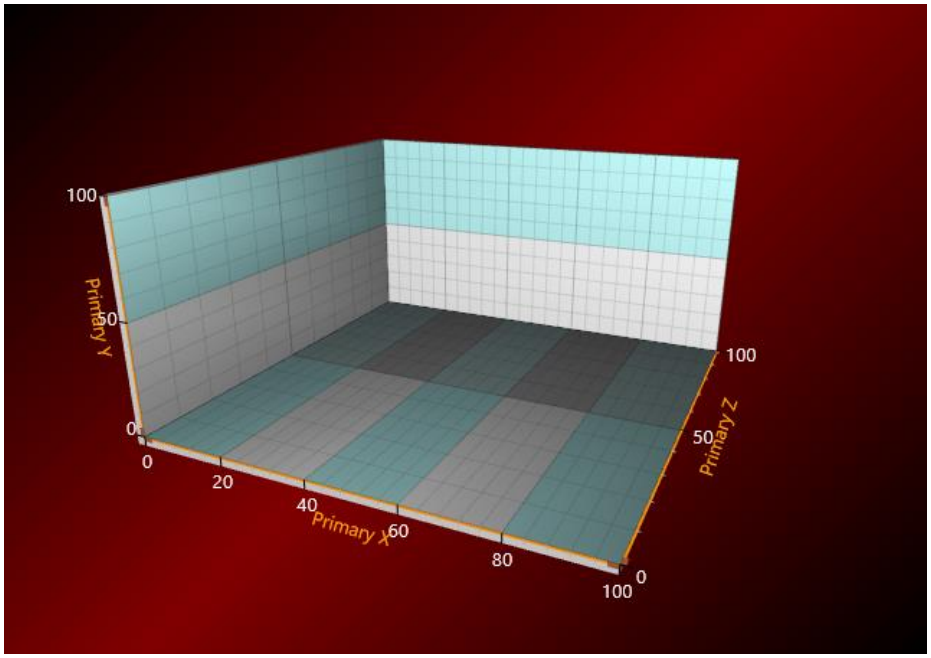


图 5-11. 在 View3D 视图模式下，设置背景（Background）为渐变圆柱形。GradientFill = Cylindrical, Color = Maroon, GradientColor = Black. GradientDirection = -45 degrees（度）

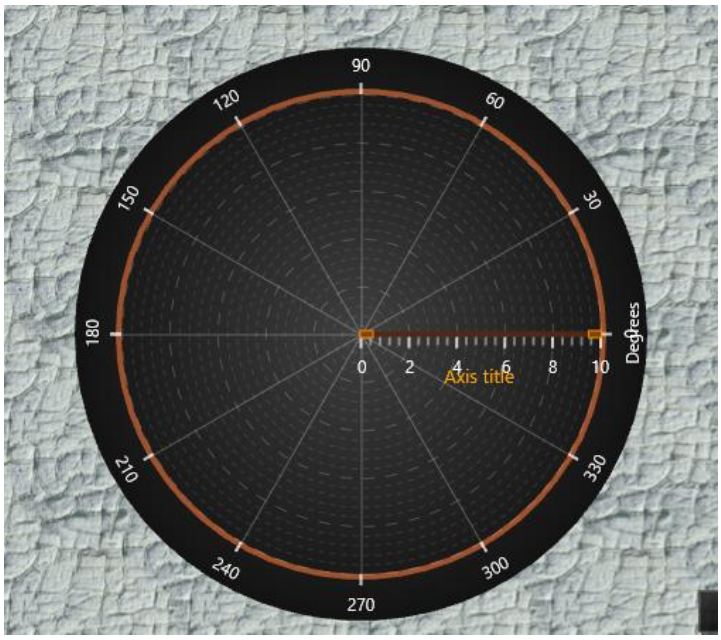


图 5-12. 在 ViewPolar 视图模式设置背景为平铺位图填充。 Style = Bitmap, 设置一张图片 Bitmap.Image, Bitmap.Layout = Tile

5.11.1 设置透明背景

在 WPF 中，可将图表设置成显示为透明，这样在图表下方的对象可以透过图表显示出来。

图 5-13. WPF 图表中的透明背景

设置 **ChartBackground.Color = #00000000**（全透明黑色）

注意！不要设置‘Transparent’（#00FFFFFF）。该设置无法穿透显示。

WinForms 不支持控件透明背景。

5.12 配置外观/性能设置

ChartRenderOptions（在 WinForms 中为 ***RenderOptions***）包含用于配置外观和性能的属性。

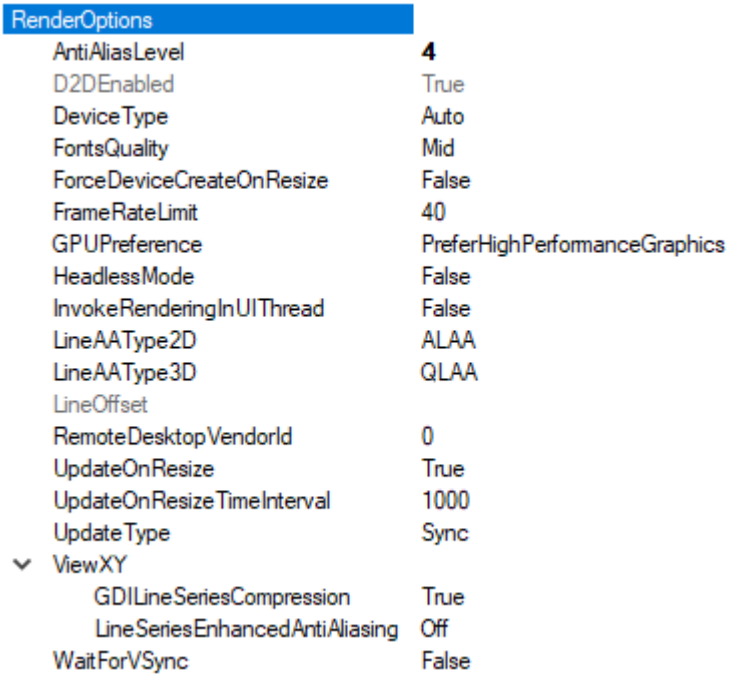


图 5-14. RenderOptions 下的属性.

DeviceType

```
// 通过代码更改渲染设备
chart.ChartRenderOptions.DeviceType = RendererDeviceType.Auto;
```

Auto 也称为 AutoPreferD11 选项，此为默认设置。

AutoPreferD9 更倾向于 DirectX9 硬件渲染，并根据可用性依照顺序：HW9 -> HW11 -> SW11 -> SW9 自动选择设备。当硬件不可用时，则会返回至 WARP（SW11）软件渲染。

AutoPreferD11 更适合 DirectX11 硬件渲染，并根据可用性依照顺序：HW11 -> HW9 -> SW11 -> SW9 自动选择设备。当硬件不可用时，则会返回至 WARP（SW11）软件渲染。**这可作为一种通用的高性能和最佳外观设置使用。** 比起采用 DirectX9 渲染程序，视觉外观会更好。

HardwareOnlyD9 仅使用 hardware 9 来进行渲染。

HardwareOnlyD11 仅使用 hardware 11 来进行渲染。

SoftwareOnlyD11 使用 DirectX11 WARP，这与 DirectX9 参考光栅设备相比非常快（但比硬件选项慢）

SoftwareOnlyD9 使用 DirectX9 参考光栅设备（非常慢）

None 如果图表被隐藏或者在背景中处于非使用状态，设置 ***DeviceType*** 为 ***None***，可以将图表资源释放给其他图表。

GPUPreference

chart.ChartRenderOptions.GPUPreference = GPUPreference.SystemSetting;

适用于带有双图形适配器系统的计算机，主要是带有集成了低性能 GPU 的 CPU/芯片组和较高性能图形 GPU（例如 AMD 或者 Nvidia）的笔记本电脑。

SystemSetting 使用在 Windows、AMD 或 Nvidia 控制面板的图形设置中可选择的选项。

PreferHighPerformanceGraphics 使用高性能 GPU（若系统中存在）。通常会带来更佳的新能表现，但能耗较高。

PreferLowPowerGraphics 使用较低性能的集成 GPU（即便系统中安装有高性能的 GPU）

默认情况下，**PreferHighPerformanceGraphics** 为首选。不要取消此项会获得最佳的性能。

FontsQuality

chart.ChartRenderOptions.FontsQuality = FontsRenderingQuality.High;

Low 性能最佳，字体边缘更光滑。仔细选择字体类型，获得最合意的外观效果。

Mid 其性能与 ***Low*** 几乎相似。字体边缘同样具有反锯齿效果。此项为默认设置。

High 外观最佳，但对性能表现有很大影响。

注意: 出于渲染技术限制，透明背景不适用于采用 ***High*** 特性设置的 DirectX 11 渲染，但可适用于 DirectX9 。

AntiAliasLevel

chart.ChartRenderOptions.AntiAliasLevel = 1;

全屏反锯齿系数。其实用性取决于硬件。值越大，外观越好，但是性能越差。设置 0 或 1 以达到性能最大化。有关反锯齿设置的更多信息，请参阅第 5.13 章节。

WaitForVSync

chart.ChartRenderOptions.WaitForVSync = true;

建议: 保持默认值。 启用后，持续渲染，直到下一次刷新（例如，1/60 秒的下一个倍数）。这只是暂时性建议，例如采用当与外部屏幕捕获应用程序同步，以防止出现条

带化，或当屏幕顶部的图像与屏幕底部不同步时。这可能显示为中断的波形数据。在启用后，特别是在 WPF 中，性能会受到明显影响。

UpdateType

```
chart.ChartRenderOptions.UpdateType = ChartUpdateTypes.Sync;
```

Sync （default）： 图表同步更新。图表在最后一次 ***EndUpdate（）*** 调用后更新，或者当设置属性（或调用一个 method）会改变图表时更新。属性的改变（没有 ***BeginUpdate…EndUpdate***）会引发即时渲染新帧。

Async: 图表以异步方式更新。图表在属性改变后将尽快更新，但是图表将在随后的某个的点渲染一个新帧。这也许让图表在某些情况下更方便使用。

LimitedFrameRate: 帧率受限于在 FrameRateLimit 属性中规定的值。0 = 无限制。例如，如果想要最大每秒刷新 10 次，则设置值为 10。这与 Async 选项类似，但会防止在第一个帧之后立即渲染新的帧，如此以来，减少帧率，但节省了系统资源。

注意! 确保在 LimitedFrameRate 和 Async 模式下正确的线程处理。如果异步更新图表，并且图表属性也同时更新，可能会引发冲突，导致图表或应用程序奔溃。

InvokeRenderingInUIThread

```
chart.ChartRenderOptions.InvokeRenderingInUIThread = true;
```

在应用程序中使用后台时，来自该线程的所有 UI 更新都必须通过 Invoke（在 WinForms 中为 ***Control.Invoke（）***，在 WPF 中为 ***Dispatcher.Invoke（）***）

在启用后，渲染部分将使用内部调用来处理 UI 线程。

默认值为 ***False***，因为以线程安全的方式设置属性和调用方法也应该要注意到，即使启用了此属性，也要防止在图表的内部状态中发生线程冲突。

HeadlessMode

```
chart.ChartRenderOptions.HeadlessMode = true;
```

将此值设置为 ***True***，可以在后台服务、控制台应用程序或其他没有用户界面的应用程序中使用图表。详情请参阅第 25 章。

5.13 DPI 处理器

DPI（每英寸点数）与显示器分辨率和尺寸有关。 高 DPI 值意味着更小的像素和更清晰的图像。

.NET Framework、.NET Desktop (.NET 5..8) 和 Windows 版本对 DPI 缩放的支持有所不同。 有关具体细节，请参阅 Microsoft 指南 (e.g. https://learn.microsoft.com/en-us/windows/win32/sbscs/application-manifests#dpiAwareness).

UWP 通常很好地支持不同 DPI 的显示器，并且可以很好地缩放应用程序内容。

}

5.13.1 概述

默认情况下，.NET Framework WPF 应用程序支持 DPI，而 WinForms 应用程序则不支持。 此外，WPF 使用 DIP（设备独立像素）代替像素来测量尺寸。 WinForms 中的默认 DPI 为 72，但值得注意的是，如果加载 wpf .dll 文件，该值将更改为 96。

NET 对 DPI 的支持与 WPF 和 WinForms 非常相似。

5.13.2 .NET Framework

当移动到具有不同 DPI 设置的另一个屏幕时，应用程序可能会也可能不会自动调整大小。 要启用应用程序调整大小，ChartOptions 下的 AllowDPIChangeInduceWindowsResize 属性需要设置为 true。 或者，用户可以注册 OnDPIChanged 事件并更改其 allowWindowResize 属性。 这些在 WinForms 中没有任何作用。

```
// Enabling automatic resizing
chart.Options.AllowDPIChangeInduceWindowResize = true;


// Via OnDPIChanged -event
chart.OnDPIChanged += chart_OnDPIChanged;

private void chart_OnDPIChanged(LightningChart chart, float dpix, float dpiy,
ref bool allowWindowResize)
{
        allowWindowResize = true;
}
```

WinForms

WinForms DPI 支持最有限，但仍然可以适应各种 DPI 设置。 应用程序可以接收 DPI 更改事件，但控件不能接收（适用于 .NET Framework 4.6.2，其他版本可能有所不同）。

可以在 app.manifest 文件中配置应用程序对 DPI 的支持。

WPF

WPF 具有更广泛的 DPI 支持。 控件还可以接收 DPI 更改事件（适用于.NET Framework 4.6.2，其他版本可能有所不同）。

可以在 app.manifest 文件中配置应用程序对 DPI 的支持。

5.13.3 .NET 6+

一般来说，NET 6+ 对 DPI 的支持相当好。 NET 应用程序的默认 DPI 感知是系统感知的，可以通过 ApplicationHighDpiMode 属性从应用程序项目文件中更改它，如下所示：

```
<PropertyGroup>
    <OutputType>WinExe</OutputType>
    <TargetFramework>net6.0-windows</TargetFramework>
    <UseWindowsForms>true</UseWindowsForms>
    <ApplicationHighDpiMode>PerMonitorV2</ApplicationHighDpiMode>
   </PropertyGroup>
```

5.14 反锯齿

LightningChart® .NET 支持反锯齿渲染。可以应用于具有 ***AntiAliasing*** 属性的对象。通过反锯齿，线条等的边缘可以渲染的更加平滑，呈现出更圆滑的图形外观，但是随着性能成本的增加，CPU/GPU 的消耗也会随之增加。

5.14.1 启动反锯齿

通过 ***AntiAliasing*** 属性可以控制反锯齿，即根据相关组件通过一个布尔值或 ***LineAntialias*** 枚举设置。对于后者而言，目前有两个可用的选项：

LineAntialias.None;	无反锯齿
LineAntialias.Normal;	反锯齿

```
seriesEventMarker.Symbol.Antialiasing = true;

pointLineSeries.LineStyle.AntiAliasing = LineAntialias.Normal;
```

图表的 ***AntiAliasLevel*** 系数也会影响反锯齿效果。这一系数基于所选的渲染引擎（DirectX9 和 DirectX11）来定义应用的反锯齿模式。设置反锯齿级别为 0 或 1，结果是渲染时不应用反锯齿，即使个别组件的 ***AntiAliasing*** 属性设置为 true 或 LineAntialias.Normal。

通过图表的渲染选项可以设置 ***AntiAliasLevel*** 值：

```
// 反锯齿系数。值0和1都不会应用反锯齿。
chart.ChartRenderOptions.AntiAliasLevel = 2;
```

没有手动设置该值时，***AntiAliasLevel*** 默认为 4。

5.14.2 DirectX 11 反锯齿

在 DirectX11 中，当用反锯齿渲染时，有几个共同的特点应该考虑到：

- 如果设置的值大于 1，那么设置 ***AntiAliasLevel*** 值会覆盖 ***AntiAliasing*** 属性，也就说，即使 ***AntiAliasing*** 属性设置为 false 或者 LineAntialias.None，也会使用反锯齿来完成渲染。唯一的例外情况是应用 LineOptimization.Hairline（只适用于使用 3D 渲染）。
- ***LineAntiAliasType*** 可用于选择是使用 alpha-blending 反锯齿（ALAA），还是使用 quadrilateral 反锯齿（QLAA）：

  LineAntiAliasingType.ALAA;Alpha-blending 反锯齿

  LineAntiAliasingType.QLAA;Quadrilateral 反锯齿

  chart.ChartRenderOptions.LineAAType2D = LineAntiAliasingType.ALAA;

RasterizerStateDescription 的 ***IsMultisampleEnabled*** 和 ***IsAntialiasedLineEnabled*** 设置还会以以下方式（只适用于渲染***线条*** line rendering）影响 QLAA 和 ALAA 渲染：

- 如果 ***RasterizerStateDescription.IsMultisampleEnabled*** == true，用 QLAA
- 如果 ***RasterizerStateDescription.IsMultisampleEnabled*** == false，用 ALAA
- 如果 ***RasterizerStateDescription.IsAntialiasedLineEnabled*** == true，用 ALAA，这只有在同时 ***RasterizerStateDescription.IsMultisampleEnabled*** == false 的时候才有效。

注意！利用 DirectX11 进行 3D 渲染时，除非设置 *AntiAliasLevel* 值为 0 或 1，否则所有三角线都总是用反锯齿渲染。

6. ViewXY

ViewXY 视图可以通过 Cartesian 坐标系、XY 图表格式，展现出各种的点线系列、区域系列、高低系列、强度系列、热图系列、柱状系列、带、线系列游标等。系列会关联到 X 和 Y 轴，使用指定的轴的值域范围。

通过 ViewXY 视图还可以显示地理地图，参阅第 6.25 章节。

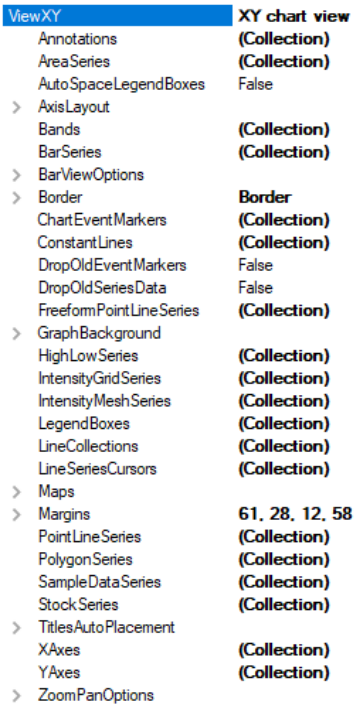


图 6-1. ViewXY 对象树

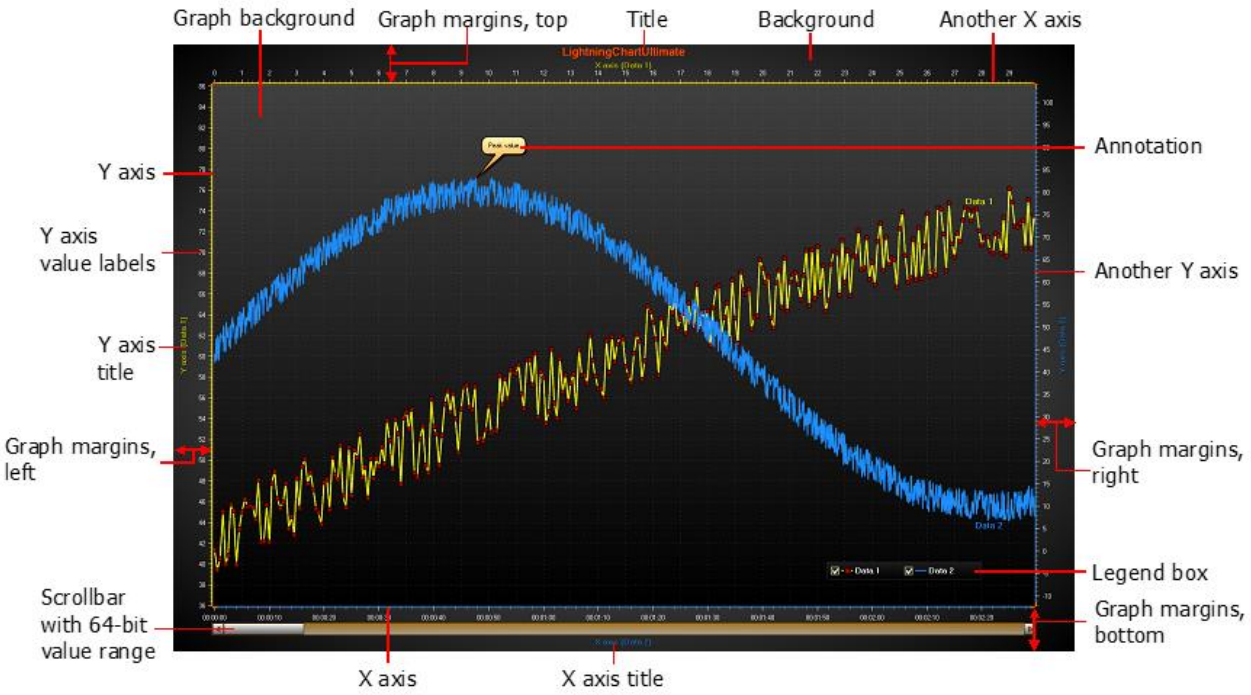


图 6-2. ViewXY 视图快速概览

Graph margins （图边距）

默认情况下，根据轴数量及其设置可自动调整图边距。通过设置 ***ViewXY.AxisLayout.AutoAdjustMargins = False***, ***Margins*** 属性应用后，可以手动设置空边的尺寸大小。将所有的图边距设置为 0，可使图形填充整个视图区域。更多详情可参阅第 6.4 章节。

Graph border （图形边框）

边框绘制在图形区域周围，边距位置上。***Border*** 属性可用以更改其颜色和可见性，以及确定是否应在系列之后渲染。关于边框更多信息可参阅第 6.4 章节。

Background （背景填充）

使用 ***Background*** （***ChartBackground in WPF***）属性来设置背景填充。其中有许多的填充选项可用，详情可参阅第 5.1110 章节。

Graph background （图背景）

使用 ***GraphBackground*** 属性来设置图形背景填充。图形是渲染所有网格、系列、系列游标、事件标记等的区域。

```
chart.ViewXY.GraphBackground.Color = Colors.DarkBlue;
```

Title （标题）

这是图表的主标题。可使用 ***Title.Text***, ***Title.Shadow***... 等属性来设置文本、阴影、颜色、文本边框、旋转、字体、对齐等。

```
chart.Title.Text = "Title text";
```

Y-axes （Y 轴）

垂直轴表示 Y 的值。参阅第 6.2 章节。

X-axis （X 轴）

水平轴表示 X 的值。参阅第 6.3 章节。

Annotations （注释）

注释可以显示在图表中与鼠标交互的文本标签或者图表。参阅第 6.26 章节。

Legend box （图例框）

其中列出图表的所有系列。参阅第 **Error! Reference source not found.**章节。

Scrollbar （滚动条）

滚动条具有 64 位无符号型取值范围，可直接支持大量的样本索引。实际上，虽然 ***HorizontalScrollBars*** 和 ***VerticalScrollBars*** 是图表根级中的集合属性，但是但他们可觉察到 ViewXY 的边界。参阅第 13 章。

6.1 轴布局选项

一般用以调整轴位置以及自动边距等的属性可以在 ***ViewXY.AxisLayout*** 属性及子属性中找到。

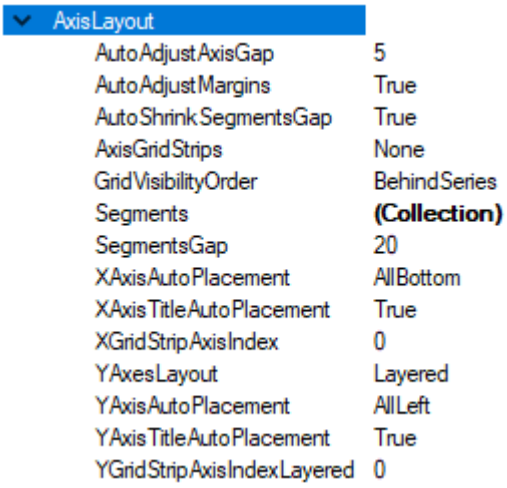


图 6-3. AxisLayout 属性树

6.1.1 设置轴位置的方法

6.1.1.1 自动布置 X 轴

演示示例：自动布置轴（Automatic axis placements）；数个轴的情况（Several axes）

XAxisAutoPlacement 控制着如何垂直放置 X 轴。

```
chart.ViewXY.AxisLayout.XAxisAutoPlacement = XAxisAutoPlacement.AllBottom;
```

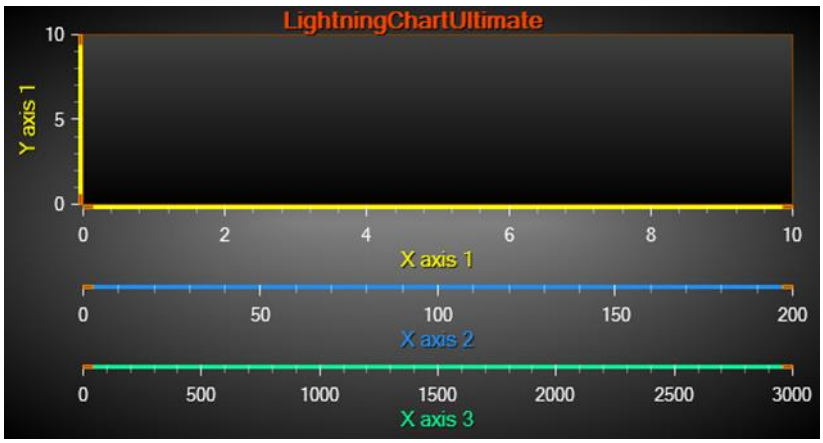


图 6-4. XAxisAutoPlacement = AllBottom；添加的三个 X 轴都置于图下方

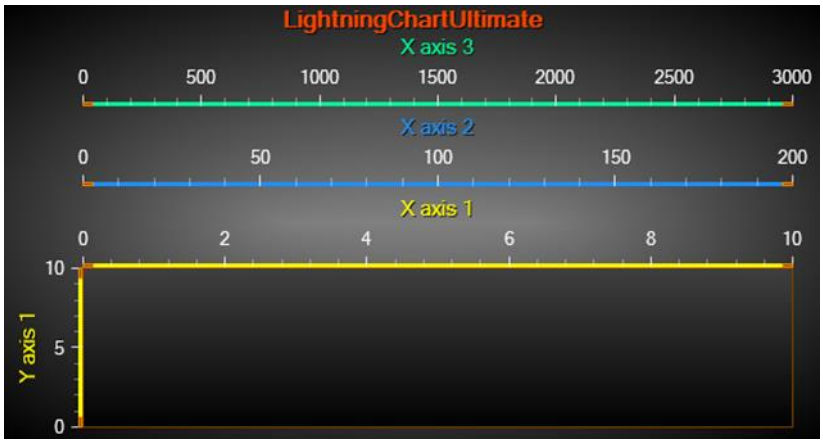


图 6-5. XAxisAutoPlacement = AllTop；所有 X 轴都置于图上方

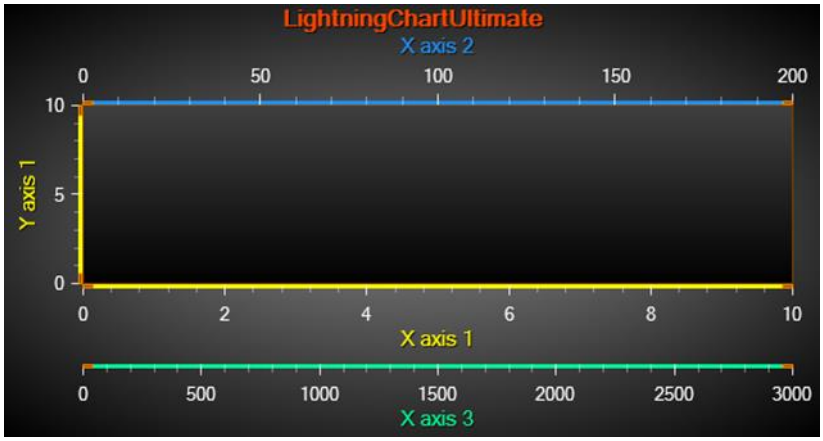


图 6-6. XAxisAutoPlacement = BottomThenTop；X 轴分散在图的上下方，从底部开始交替分散在两边。

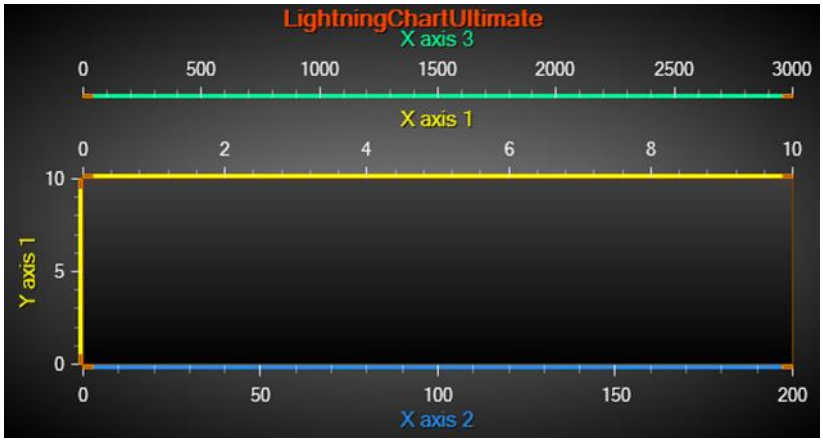


图 6-7. XAxisAutoPlacement = TopThenBottolm；轴分布在图的上下方，从顶部开始交替分散在两边。

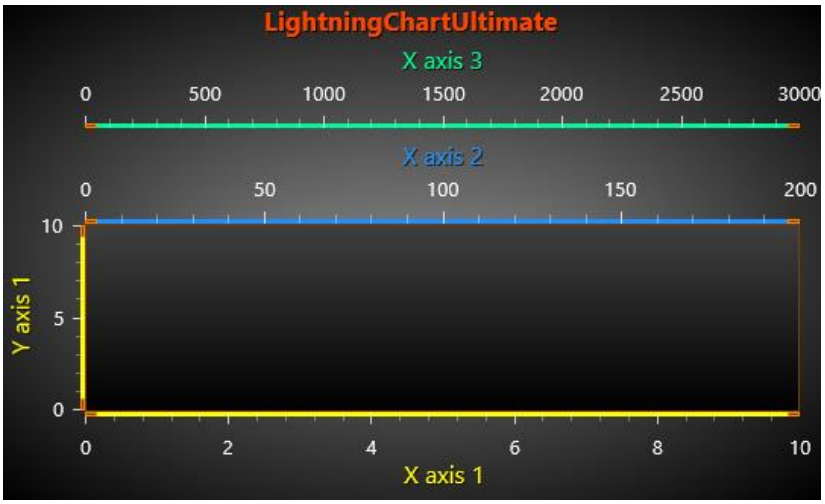


图 6-8. XAxisAutoPlacement = Explicit；X 轴出现在显式选择的那一侧。 X 轴 1（XAxis1）的 ExplicitAutoPlacementSide 属性设置为底部（Bottom），而 X 轴 2 和 3（XAxis2 和 XAxis3）为顶部（Top）。

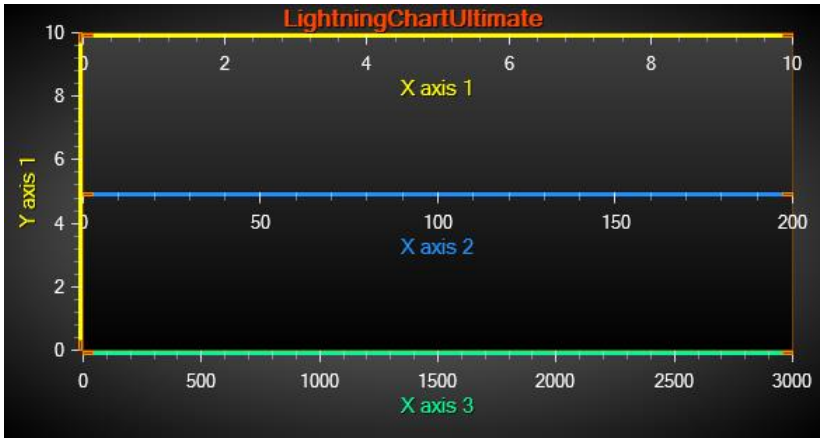


图 6-9. XAxisAutoPlacement = Off；禁用自动布置轴，每个轴的位置（Position）和对齐（Alignment）属性各自分别应用。第一条轴 Position = 0，第二条轴 Position = 50，第三条轴 Position = 100

6.1.1.2 自动布置 Y 轴

演示实例：Y 轴布局（Y axis layouts*）；自动布置轴（Automatic axis placements）；数个轴的情况（Several axes）*

YAxisAutoPlacement 控制着如何水平放置 Y 轴。

```
chart.ViewXY.AxisLayout.YAxisAutoPlacement = YAxisAutoPlacement.AllLeft;
```

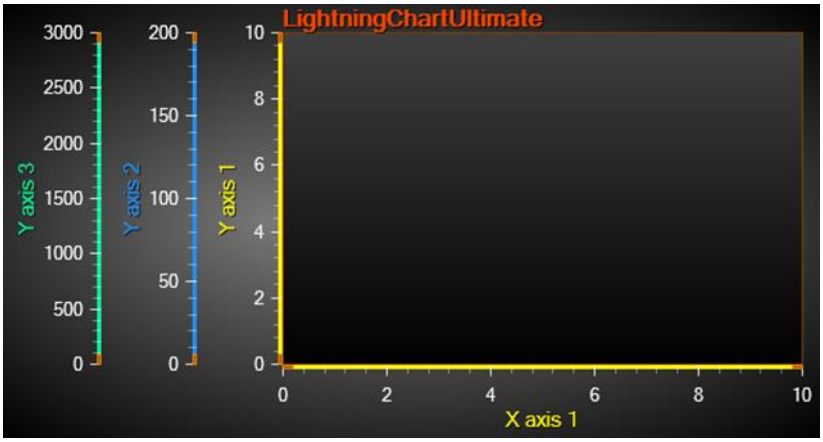


图 6-10. YAxisAutoPlacement = AllLeft；添加的三个 Y 轴都置于图左侧。

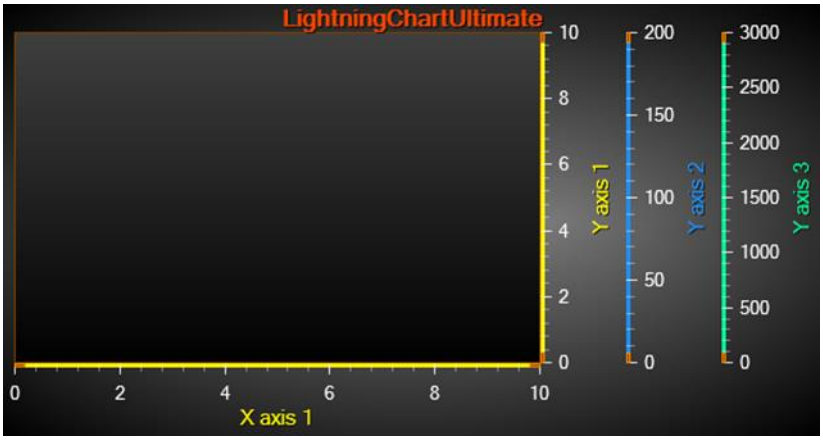


图 6-11. YAxisAutoPlacement = AllRight；所有 Y 轴都置于图右侧。

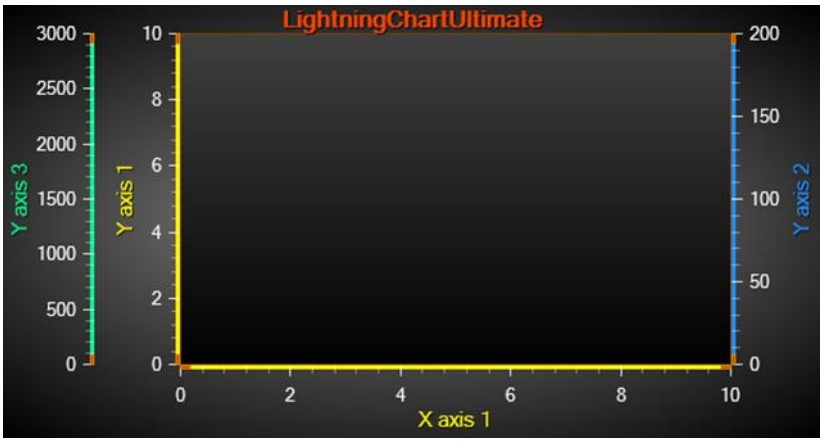


图 6-12. YAxisAutoPlacement = LeftThenRight；Y 轴分布在图左右两侧，从左侧开始交替分散在两侧。

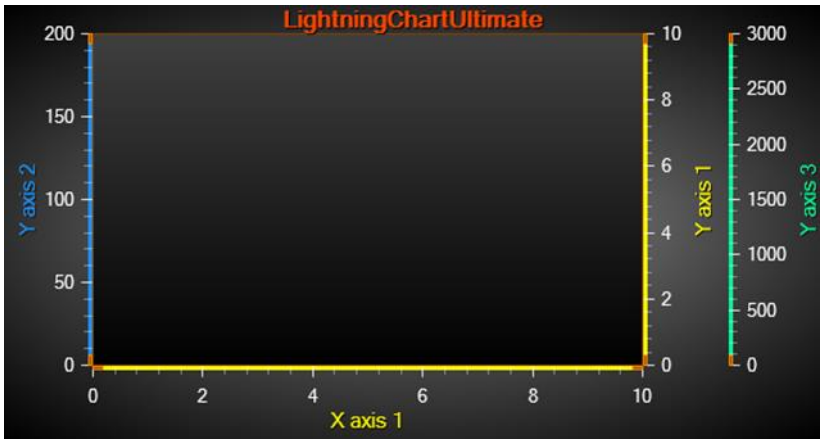


图 6-13. YAxisAutoPlacement = RightThenLeft；Y 轴分布在图左右两侧，从右侧开始交替分散在两侧。

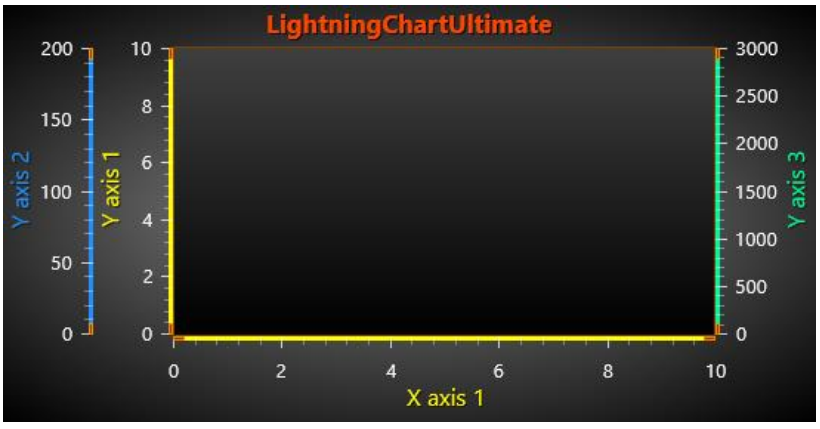


图 6-14. YAxisAutoPlacement = Explicit；Y 轴出现在显式选择的那一侧。Y 轴 1 和 2（YAxis1 和 YAxis2）的 ExplicitAutoPlacementSide 属性设置为左侧（Left），而 Y 轴 3 为右侧（Right）。

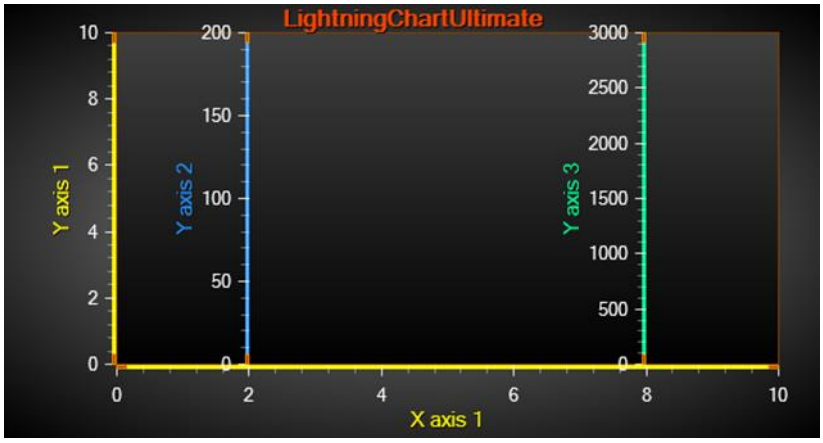


图 6-15. YAxisAutoPlacement = Off；禁用自动布置轴，每个轴的位置（Position）和对齐（Alignment）属性各自分别应用。第一条轴 Position = 0，第二条轴 Position = 20，第三条轴 Position = 80

6.1.2 图形段（Graph segments）及其中的 Y 轴位置

如果定义有数个 Y 轴，它们可以以三种不同的方式垂直排列： ***Layered（层叠式）***、***Stacked（叠置式）*** 和 ***Segmented（分段式）***。通过 ***ViewXY.AxisLayout.YAxesLayout*** 属性可以对其进行选择。

6.1.2.1 *层叠式（Layered）*

演示实例：Y 轴布局（Y axis layouts）；自动布置轴（Automatic axis placements）

在 ***Layered*** 视图模式中，所有的 Y 轴都是从图的顶部开始，延伸到图的底部。Y 轴和与其关联的系列在同一个垂直空间里。

```
chart.ViewXY.AxisLayout.YAxesLayout = YAxesLayout.Layered;
```

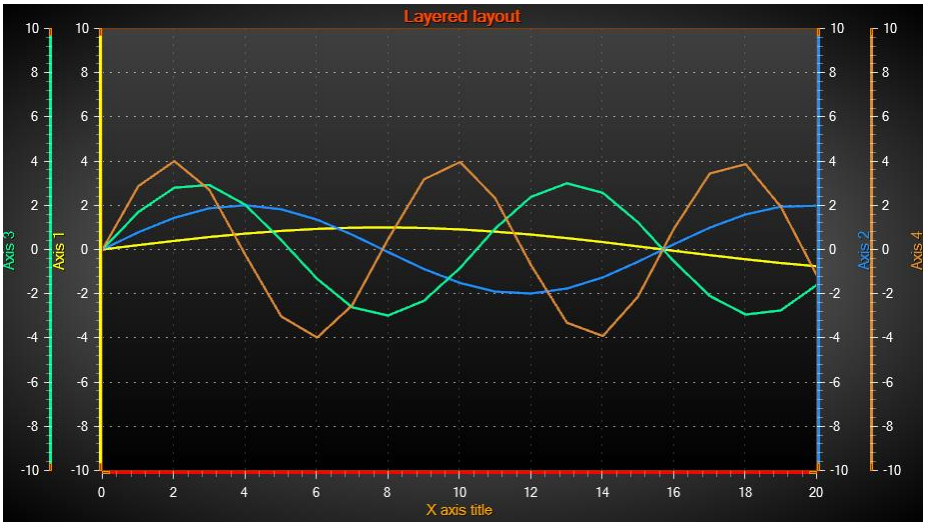


图 6-16. 4 个 Y 轴设定为 YAxesLayout = Layered 时的示例视图

6.1.2.2 *叠置式（Stacked ）*

演示实例：Y 轴布局（Y axis layouts）；多信道游标追踪（Multi-channel cursor tracking）；系列中数据中断（Data breaking in series）

在 ***Stacked*** 视图中，每条 Y 轴都有自己的垂直空间。所有 Y 轴都等高。

```
chart.ViewXY.AxisLayout.YAxesLayout = YAxesLayout.Stacked;
```

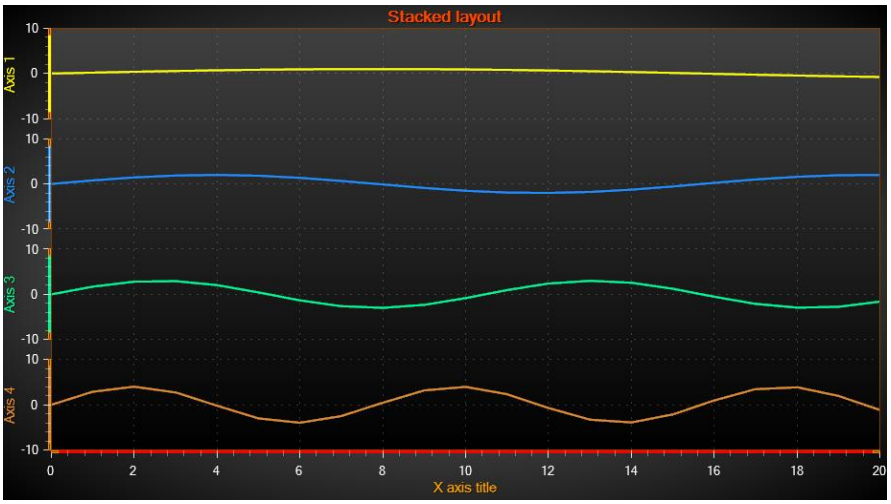


图 6-17. 4 条 Y 轴在设置为 YAxesLayout = Stacked 时的示例视图

6.1.2.3 *分段式（Segmented）*

演示示例：Y 轴布局（Y axis layouts）；多图例情况（Multiple legends）；分割器分段（Segments with splitters）

在 ***Segmented*** 视图中，垂直空间分割成了数个段（***Segments***）。每个段都可以包含数个 Y 轴。每个段的相对高度可以设置，同时，一个段内的所有 Y 轴都具有与该段一样的高度。

```
chart.ViewXY.AxisLayout.YAxesLayout = YAxesLayout.Segmented;
```

Segments 必须在 ***AxisLayout.Segments*** 集合中创建。首先添加的段要置于图表的底部。一个段只有一个属性，即 ***Height***；它是一个相对于其他段的相对高度。因为段的比例需要随着图表的尺寸来调整，所以段的高度不是以屏幕像素来规定的。

```
// 添加2个段，第二个段的高度是第一个段的2倍
chart.ViewXY.AxisLayout.Segments.Add（new YAxisSegment（））;
chart.ViewXY.AxisLayout.Segments.Add（new YAxisSegment（））;
chart.ViewXY.AxisLayout.Segments[0].Height = 1;
chart.ViewXY.AxisLayout.Segments[0].Height = 2;
```

通过设置 ***yAxis.SegmentIndex*** 属性可以为 Y 轴分配一个段。***SegmentIndex*** 则为 ***AxisLayout.Segments*** 集合中的索引。

```
chart.ViewXY.YAxes[0].SegmentIndex = 0;
chart.ViewXY.YAxes[1].SegmentIndex = 1;
```

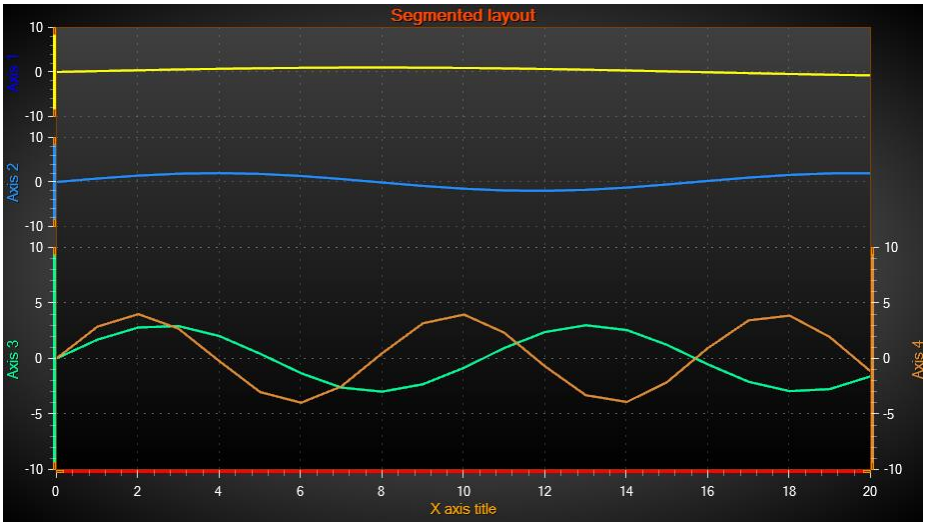


图 6-18. 4 个 Y 轴在设置为 YAxesLayout = Segmented 时的示例视图。前两个段的高度设置为 Height = 1，最后一个段的高度为 2.5。 Axis1.SegmentIndex = 0, Axis2.SegmentIndex = 1, Axis3 与 Axis4.SegmentIndex = 3.

当选择 ***Stacked*** 或 ***Segmented*** 视图后，利用 ***ViewXY.AxisLayout.SegmentsGap*** 属性可以调整各图形段之间的垂直距离。

```
chart.ViewXY.AxisLayout.SegmentsGap = 10; // 每个段之间设置 10 像素间距
```

如果定义了许多的 Y 轴，应启用 ***AutoShrinkSegmentsGap*** 属性来自动缩减间距。这样，在绘制每条 Y 轴时至少都会有一些垂直距离。

```
chart.ViewXY.AxisLayout.AutoShrinkSegmentsGap = false;
```

如果需要实现段特定的用户界面逻辑，使用 ***ViewXY.GetGraphSegmentInfo（）***–method 能找到图形段的边界。

```
// 获取每个段的顶部与底部坐标
float[] topCoords = chart.ViewXY.GetGraphSegmentInfo（）.SegmentTops;
float[] bottomCoords = chart.ViewXY.GetGraphSegmentInfo（）.SegmentBottoms;
```

6.1.3 轴网格带（Axis grid strips）

演示示例：历史数据回顾（Historic data review）；缩放条图表（Zoom bar chart）

轴网格（划分）间隔可以在图形背景上显示为填充效果。通过 ***ViewXY.AxisLayout.AxisGridStrips*** 对 ***X*** 进行设置，可在 X 轴设置网格带。另外，通过 ***AxisGridStrips*** 对 ***Y*** 进行设置，可在 Y 轴设置网格带。用 ***Both*** 选项可以对 X 和 Y 轴都设置网格带，而用 ***None*** 则可不显示任何网格带。

```
chart.ViewXY.AxisLayout.AxisGridStrips = XYAxisGridStrips.X;
```

当使用多条 X 轴时，可以使用 ***XGridStripAxisIndex*** 对 X 轴设置网格带。不过，一次只能对一条 X 轴进行设置。

chart.ViewXY.AxisLayout.XGridStripAxisIndex = 0;

当使用 ***Layered YAxisLayout*** 选项时，可以通过 ***YGridStripAxisIndexLayered*** 对 Y 轴设置网格带。如果是 ***Stacked*** 布局，每条 Y 轴都具有网格带。

chart.ViewXY.AxisLayout.YGridStripAxisIndexLayered = 0;

在 X 轴或 Y 轴对象的 ***GridStripColor*** 属性中，可以对网格带的颜色进行调整。

chart.ViewXY.YAxes[0].GridStripColor = Color.FromArgb（80, 0, 0, 100）;

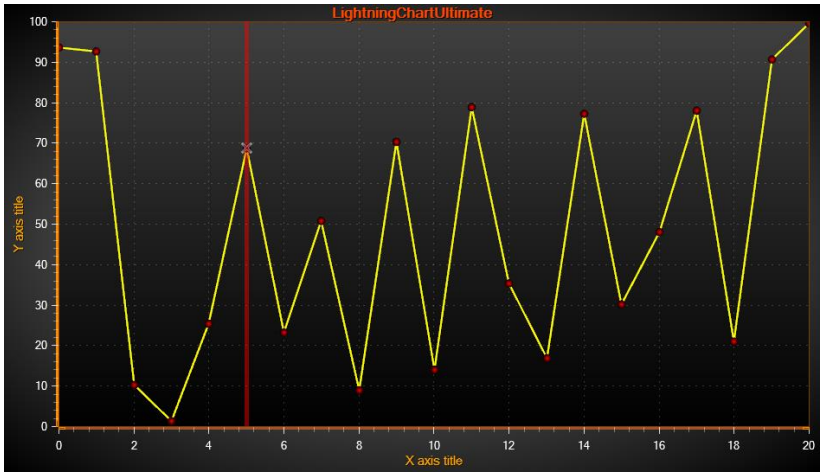


图 6-19. AxisGridStrips = None.

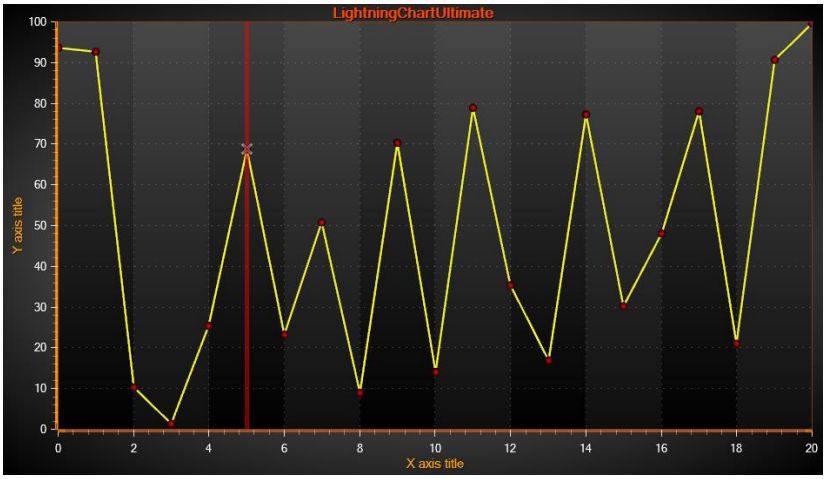


图 6-20. AxisGridStrips =X.

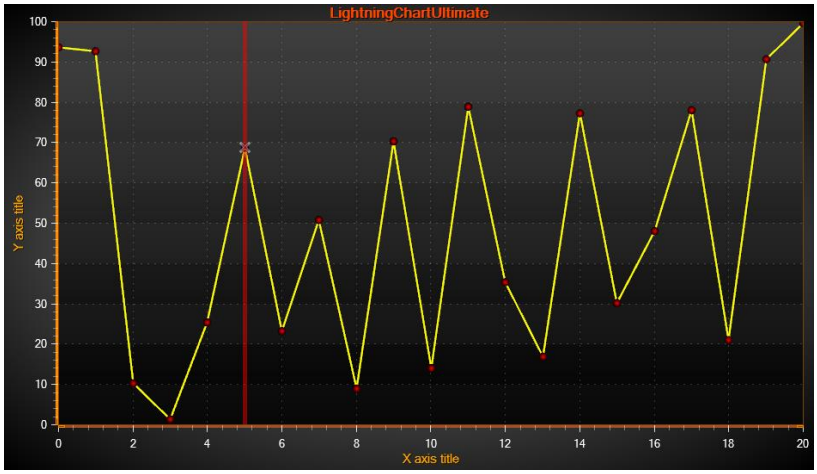


图 6-21. AxisGridStrips = Y.

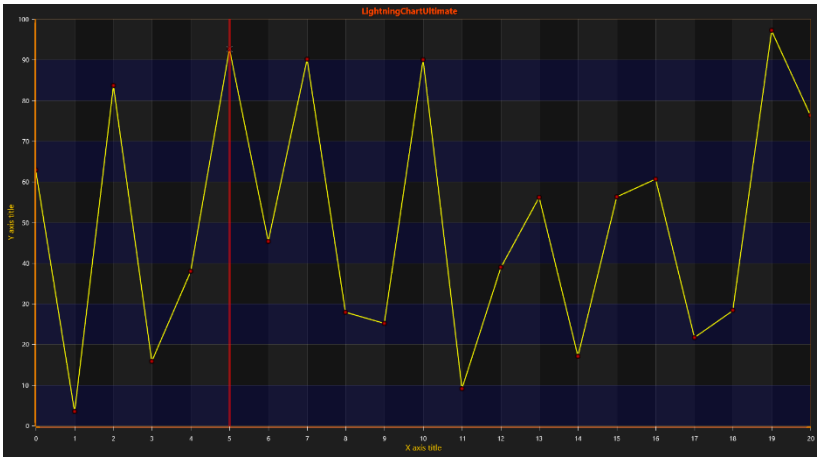


图 6-22. AxisGridStrips = Both. GridStripColor 同样在 Y 轴上也发生了变化。

6.1.4 将 Y 值限制到堆栈段

每个 XY 系列的 **LimitYToStackSegment** 属性。 启用后，系列将被剪裁到 segment 及其所属的 Y 轴区域。 在大多数情况下，该属性是一个布尔值来控制是否数据是否应该被裁剪。 然而，一些较新的系列（**SampleDataBlockSeries**、**LiteLineSeries** 和 **LiteFreeformLineSeries**) 有额外的选项。 还有 **LimitYToStackSegment** 是枚举的带选项的类型：None（无剪裁）、Clip（线将像旧系列一样被剪裁）和 ClampToSegment（如果线超过线段的边缘，它将在边缘水平呈现）。 请注意，对于 **LiteFreeformLineSeries**，启用 ClampToSegment 的线将沿着边缘呈现，直到真正的点 X 值是。 因此，它可能比其他系列的夹线更长。

6.1.5 其他 AxisLayout 选项

启用 ***XAxisAutoPlacement*** 或 ***YAxisAutoPlacement*** 后，用 ***AutoAdjustAxisGap*** 可以设置两个相邻轴区域之间的间距（以像素为单位）。

```
chart.ViewXY.AxisLayout.XAxisAutoPlacement = XAxisAutoPlacement.AllBottom;
chart.ViewXY.AxisLayout.AutoAdjustAxisGap = 10;
```

启用 ***XAxisTitleAutoPlacement*** （或 ***YAxisTitleAutoPlacement***）后，轴的标题距离根据数值标签的长度、轴和刻度线的对齐选项自动计算。如果禁用了 ***XAxisTitleAutoPlacement*** （或 ***YAxisTitleAutoPlacement***），则用轴对象属性的 ***Title.DistanceToAxis*** 设置标题到轴线的距离。

```
chart.ViewXY.AxisLayout.XAxisTitleAutoPlacement = false;
chart.ViewXY.XAxis[0].Title.DistanceToAxis = -20;
```

6.2 Y 轴

Y 轴的数量可以定义无限多个。可以使用 ***YAxes*** 集合属性来添加 Y 轴。

```
// 向图表添加Y轴
chart.ViewXY.YAxes.Add（new AxisY（））;

AxisY axisY = new AxisY（_chart.ViewXY）;
axisY.Title.Text = "Y-axis";
chart.ViewXY.YAxes.Add（axisY）;
```

6.2.1 Y 轴类属性

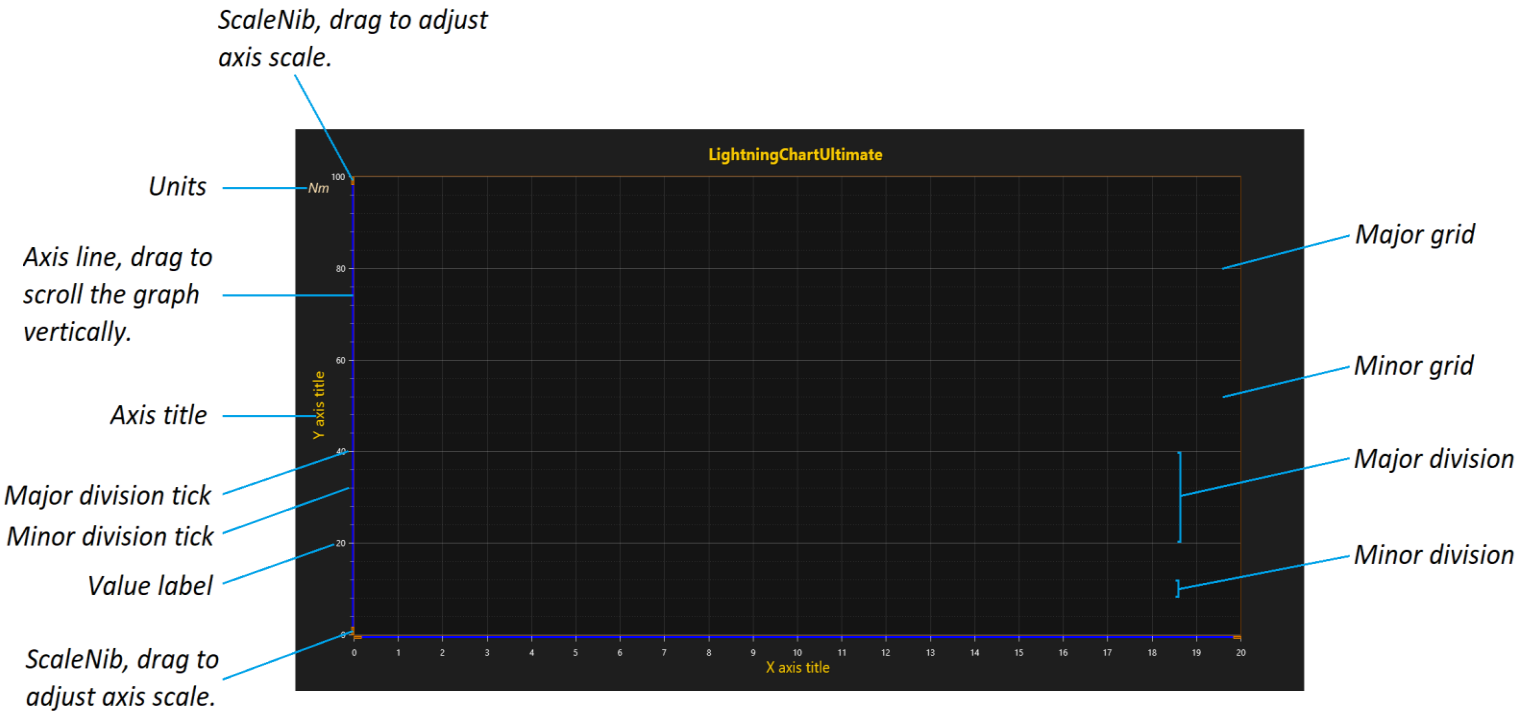


图 6-23. Y 轴，各主要分度及网格

6.2.2 刻度值标签格式化

演示示例：（High-Low）； 温度图（Temperature graph）；多信道游标追踪（Multi-channel cursor tracking）； 地图路线（Map route）

设置 ***AutoFormatLabels*** 可以对小数计数、时间格式、或使用指数形式，进行自动计算，以适应可视范围。若要手动设置数值格式，应禁用 ***AutoFormatLabels***。

```
chart.ViewXY.YAxes[0].AutoFormatLabels = false;
```

用 ***LabelsNumberFormat*** 可以设置数值的格式。

```
// 总采用两位小数
chart.ViewXY.YAxes[0].LabelsNumberFormat = "0.00";

// 用一位小数表示指数
chart.ViewXY.YAxes[0].LabelsNumberFormat = "0.0E+00";
```

利用 ***LabelsTimeFormat*** 属性，可手动来设置时间格式。这支持任何秒分数的计数（例如“.fffffff”），可以精确的缩放视图。

```
// 显示小时、分钟、秒钟以及四位有效数字
_chart.ViewXY.YAxes[0].LabelsTimeFormat = "HH:mm.ss.ffff";
```

6.2.3 值类型

ValueType –属性控制着轴标签使用的是哪种值类型。

```
// 更改轴的值类型
chart.ViewXY.YAxes[0].ValueType = AxisValueType.DateTime;
```

ValueType 中具有以下几个选项：

Number

表示整数及分数时常用的数字格式。禁用 ***AutoFormatLabels***，会应用 ***LabelsNumberFormat***。

Time

表示时间。禁用 ***AutoFormatLabels***，会应用 ***LabelsTimeFormat***。

DateTime

表示日期，可选时间。此选项与格式 ***Time*** 类似，禁用 ***AutoFormatLabels*** ，也会应用 ***LabelsTimeFormat***。

注意！为达到最佳精确度，建议在图表中显示的日期下方设置 ***DateOriginYear***, ***DateOriginMonth*** 和 ***DateOriginDay***。使用 ***DateTimeToAxisValue*** 方法，从用于系列数据的一个.NET ***DateTime*** 对象中获取轴值。

```
// 将当前时间转换为 Y 值
data[0].Y = chart.ViewXY.YAxes[0].DateTimeToAxisValue（DateTime.Now）;
```

MapCoordsDegrees

用十进制度数表示地图坐标。
示例：　　*40.446195° -79.948862°*

MapCoordsDegNESW

用十进制度数表示地图坐标，附带 N、E、S、W 指示方向。
示例：　　*40.446195N　79.948862W*

MapCoordsDegMinSecNESW

用度-弧分-弧秒表示地图坐标，附带 N、E、S、W 指示方向。
示例：　　*40°2'13"N　9°58'2"W*

MapCoordsDegPadMinSecNESW

用度-弧分-弧秒表示地图坐标，附带 N、E、S、W 指示方向。若弧分和弧秒的值< 10，则用零填补位数。这种方法可以对齐数字，所以是在 Y 轴上呈现坐标的非常好的方法。
示例：　　*40°02'13"N　9°58'02"W*

6.2.4　值域设置

通过向 ***Minimum*** 和 ***Maximum*** 属性赋值来设置一条轴的值域范围。***Minimum*** 应该小于 ***Maximum*** 。如果试图设置 ***Minimum*** > ***Maximum*** 时，或者相反，那么内部限制器将限制该值近似于另一个值。若要同时设置这两个值，可以使用 ***SetRange（…）*** method 。在 ***SetRange*** 中若 ***Minimum*** > ***Maximum*** ，会自动翻转数值，以满足 ***Minimum*** < ***Maximum*** 。

```
chart.ViewXY.YAxes[0].Minimum = 5;
chart.ViewXY.YAxes[0].SetRange（5, 10）;
```

当启用 ***MouseScrolling*** 后，用鼠标拖动 Y 轴，可以直接滚动 Y 轴的值域范围。启用 ***MouseScaling*** 属性后，可以通过向上或向下拖动比例尺的尖端区域（轴的末端）来修改 ***Minimum*** 或 ***Maximum*** 。

```
// 启用/禁用鼠标拖动
chart.ViewXY.YAxes[0].AllowScrolling = true;
chart.ViewXY.YAxes[0].AllowScaling = true;
```

6.2.5　恢复值域范围

当从右向左应用鼠标缩放时，可以使用轴的属性 ***RangeRevertEnabled***、***RangeRevertMaximum*** 和 ***RangeRevertMinimum*** 将轴范围恢复到特定的值。详情可参阅第 6.28.5 章节。

6.2.6 分度

分度由 ***MajorDiv*** 和 ***MinorDiv*** 属性控制，可确定图表中主要和次要刻度的数量。例如，设置 5 个主要分度可以把 Y 轴划分为五个大小相等的间隔，这些间隔间由一个刻度和一条主网格线分隔。默认情况下，主刻度处于启用状态，次刻度禁用。

```
// 启用主分度刻度
chart.ViewXY.YAxes[0].MinorDivTickStyle.Visible = true;
```

AutoDivSpacing –该属性可自动计算主要分度。默认为启用状态。间距根据值标签的字体大小和 ***AutoDivSeparationPercent*** 属性计算，尽可能做到人性化。设置 ***AutoDivSeparationPercent*** 可在值标签之间留下一定空间，但由于是基于标签的高度，这意味着增大值会减少主要分度的数量。***AutoDivSpacing*** 和 ***AutoDivSeparationPercent*** 不会影响次分度，反之，它们是根据 ***MinorDivCount*** 属性值在主要分度之间计算的。

```
//每个标签之间留出值标签高度的 100%
chart.ViewXY.YAxes[0].AutoDivSeparationPercent = 100;
```

如果禁用 ***AutoDivSpacing*** ，分度间距可用 ***MajorDiv*** 和 ***MajorDivCount*** 属性进行手动控制。***MajorDiv*** 通过大小控制间距，而 ***MajorDivCount*** 通过分度数量进行控制。无论 ***MajorDiv*** 如何设置，当轴的范围发生变化时，可以使用 ***KeepDivCountOnRangeChange*** 属性强制保持相同的分度数量。

```
// 以 20 为单位的主刻度 （0, 20, 40, 60…）
chart.ViewXY.YAxes[0].MajorDiv = 20;
// 准确显示 5 个主分度
chart.ViewXY.YAxes[0].MajorDivCount = 5;
//当轴的范围发生变化时，保持相同的分度数量
chart.ViewXY.YAxes[0].KeepDivCountOnRangeChange = true;
```

通过 ***MajorDivTickStyle*** 属性可以设置主分度的刻度样式。使用 ***MajorDivTickStyle.Alignment*** 属性可编辑刻度和标签的方向。将值标签绘制在主分度刻度旁边。另外，用 ***MinorDivTickStyle*** 属性可以修改次分度属性。

```
// 更改主分度刻度的的对齐方式和刻度长度
chart.ViewXY.YAxes[0].MajorDivTickStyle.Alignment = Alignment.Far;
chart.ViewXY.YAxes[0].MajorDivTickStyle.LineLength = 20;
```

6.2.7 网格

在分度刻度的垂直位置上绘制水平网格线。主分度刻度为主网格，次分度刻度为次网格。***MajorGrid*** 和 ***MinorGrid*** 属性可用于编辑网格外观。

```
// 修改网格样式
chart.ViewXY.YAxes[0].MajorGrid.Color = Color.FromArgb（100, 200, 200, 200）;
chart.ViewXY.YAxes[0].MinorGrid.Pattern = LinePattern.Dash;
```

6.2.8 自定义刻度

演示示例：自定义轴刻度（Custom axis ticks）；日期轴和自动以刻度（Date axes and custom ticks）

使用自定义刻度可以对轴刻度位置和标签文本进行手动设置。设置 ***CustomTicksEnabled*** 为 true，并用 ***CustomTicks*** 属性列表定义刻度位置。

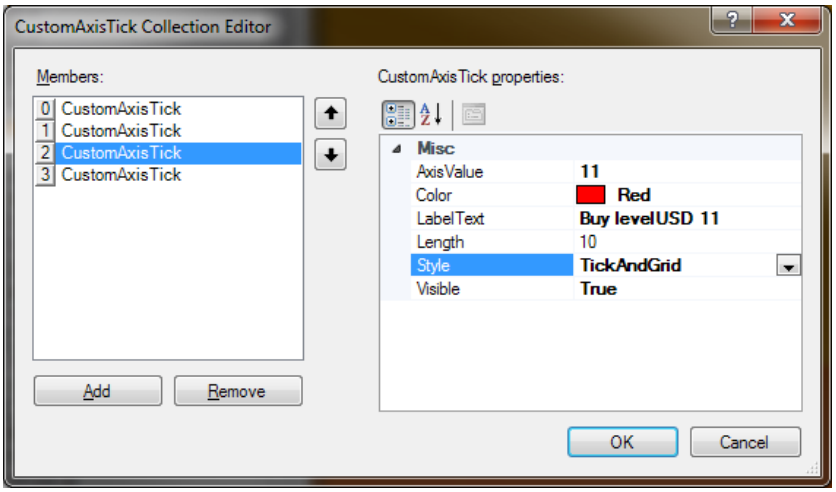


图 6-24. 自定义刻度属性

自定义刻度可以由刻度、网格或两者组成。使用 ***Style*** 可分别在 Tick、Grid 和 TickAndGrid 之间进行选择。通过 ***Color*** 属性可以对刻度和网格的颜色进行修改。在 ***Length*** 属性中可设置刻度长度。在轴的 ***MajorGrid.Pattern*** 和 ***PatternScale*** 属性中可以设置网格线的式样。

CustomAxisTick 有 ***AxisValue*** 和 ***LabelText*** 两个属性，可以定义位置和相应的标签文本。在使用自定义刻度时，禁用 ***AutoFormatLabels*** 可以显示自定义标签文本。另外，在通过代码设置新的自定义刻度后应调用 ***InvalidateCustomTicks（）***。

```
// 添加一自定义刻度为绿色刻度和网格线
_chart.ViewXY.YAxes[0].CustomTicks.Add（new CustomAxisTick（_chart.ViewXY.YAxes[0],  14, "Sell level\nUSD 14", 10, true, Colors.Green, CustomTickStyle.TickAndGrid））；

// 白色刻度，无网格线
_chart.ViewXY.YAxes[0].CustomTicks.Add（new CustomAxisTick（_chart.ViewXY.YAxes[0], 12.2, "Month\nmedian", 20, true, Colors.White, CustomTickStyle.Tick））；

// 红色刻度和网格线
_chart.ViewXY.YAxes[0].CustomTicks.Add（new CustomAxisTick（_chart.ViewXY.YAxes[0], 11, "Buy level\nUSD 11", 10, true, Colors.Red, CustomTickStyle.TickAndGrid））；

// 显示自定义刻度字符串
_chart.ViewXY.YAxes[0].CustomTicksEnabled = true;
_chart.ViewXY.YAxes[0].AutoFormatLabels = false;
_chart.ViewXY.YAxes[0].MajorGrid.Pattern = LinePattern.Dot;
_chart.ViewXY.YAxes[0].InvalidateCustomTicks（）；
```

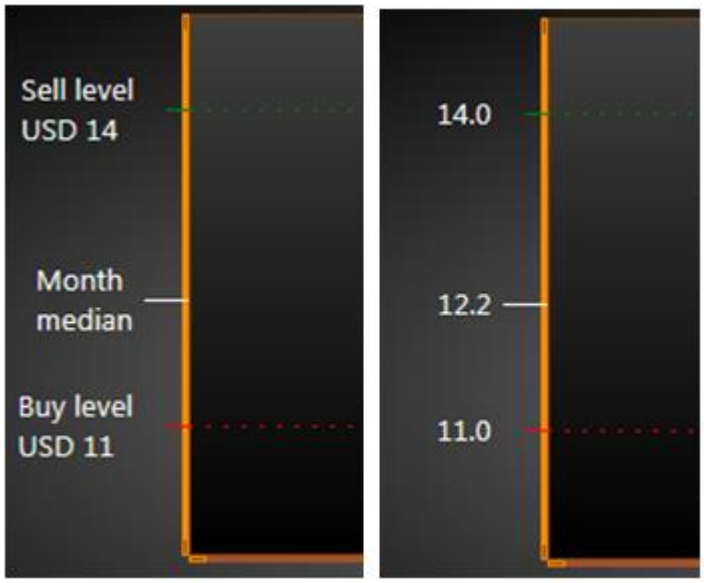


图 6-25. Y 轴上的自定义刻度；左侧，axis.AutoFormatLabels = false；右侧，AutoFormatLabels = True.

当设置 ***CustomAxisTicksEnabled*** 为 true 时，不显示次刻度或网格。要设置任意的次刻度或网格，只需在 ***CustomTicks*** 集合中添加 ***CustomAxisTicks***，并设定颜色或线长度。

6.2.9 基于事件的轴值格式化

演示示例： Axis range edit, value labels; Business dashboard; Intensity persistent layer, signal

除去 ***CustomAxisTicks*** 之外，还可以通过 ***FormatValueLabel*** 事件来对轴的值标签进行格式化。它根据返回的字符串值，来修改对应的轴的每个值标签。事件的 ***e.Axis*** 和 ***e.Value*** 属性，可用于访问被修改的轴对象和标签值。但是和 ***CustomAxisTicks*** 不同的是，***FormatValueLabel*** 不能用以更改标签的位置，因为轴的分度设置（参阅第 6.2.6 章节）仍然限定着位置。

```
// 对 Y 轴设置订阅 FormatValueLabel 事件
_chart.ViewXY.YAxes[0].FormatValueLabel += Chart_FormatValueLabel;

// 在事件内修改值标签
private string Chart_FormatValueLabel（object sender, FormatValueLabelEventArgs e）
{
    return "Y-axis value: " + e.Value.ToString（）;
}
```

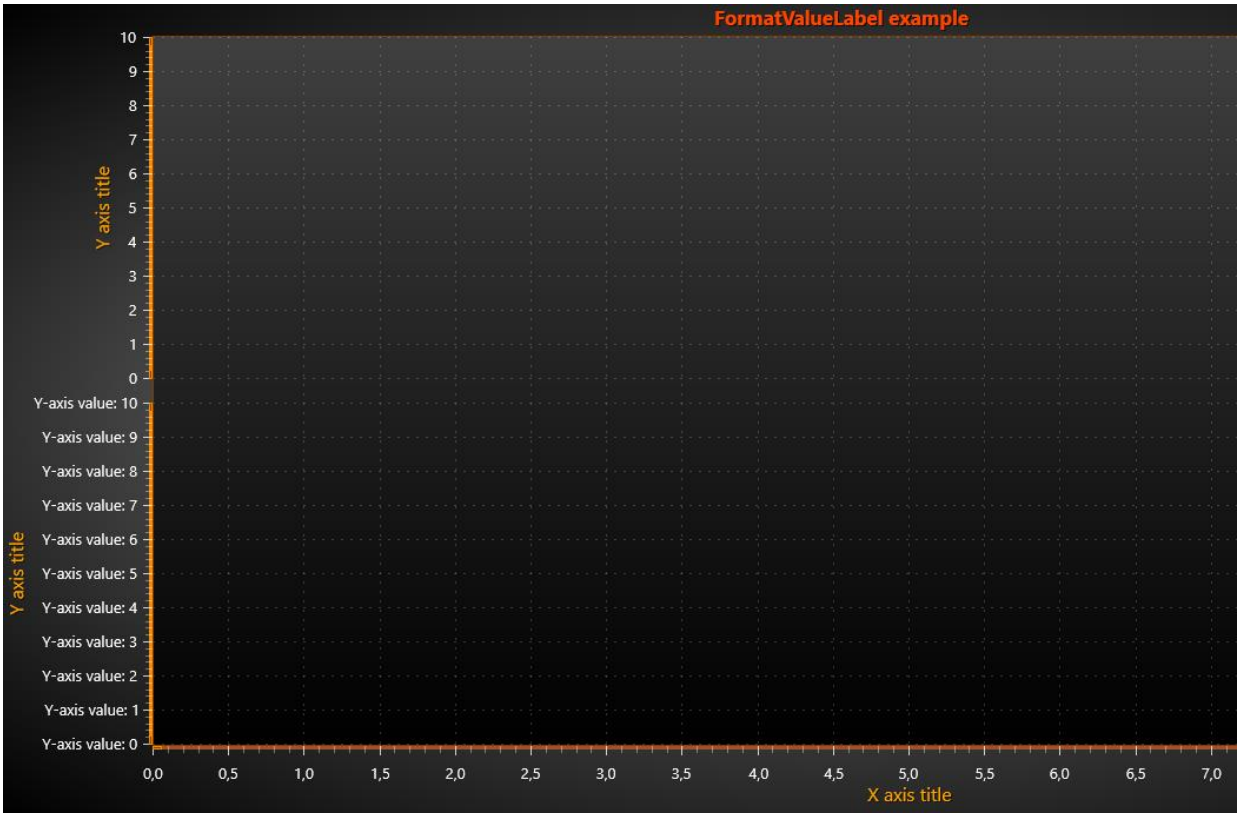


图 6-26. 对下方的 Y 轴设置 FormatAxisValues； 值显示为"Y-axis value: " + 当前值

FormatValueLabel 可用于 Y 轴和 X 轴，以及在 View3D 视图中的每条轴。

6.2.10 X 轴与 Y 轴翻转

X 轴与 Y 轴可以翻转显示，这样的话，最小值会高于/大于最大值。举例来说，当要从外观上消除分配给 Y 轴的系列数据极性时，这是一个非常方便的功能。

```
chart.ViewXY.YAxes[0].Reversed = true;
```

6.2.11 对数轴

演示示例：对数轴（Logarithmic axes）；最小对数值（Minimal logarithmic values）；Log axis fit, ignore zeros

将 ***ScaleType*** 设置为 ***Logarithmic***，可使用对数显示。使用 ***LogBase*** 属性设置对数的底。图表还可以显示 0…1 之间的对数值。使用 ***LogZeroClamp*** 设置在轴中设置最小值。若要使用对数轴的典型最小值，则设置为 1。若要使用低于 0 的值，则适当设置一个小的正数值，如 1.0E-20，最好适合所使用的数据。另外，设置 ***LogLabelsType*** 可对刻度标签使用特殊格式。

```
// 设置对数轴
chart.ViewXY.YAxes[0].ScaleType = ScaleType.Logarithmic;
chart.ViewXY.YAxes[0].LogBase = 10;
chart.ViewXY.YAxes[0].LogZeroClamp = 1;
chart.ViewXY.YAxes[0].LogLabelsType = LogLabelsType.Log10Exponential;
```

6.2.11.1 以 10 为底的指数表示

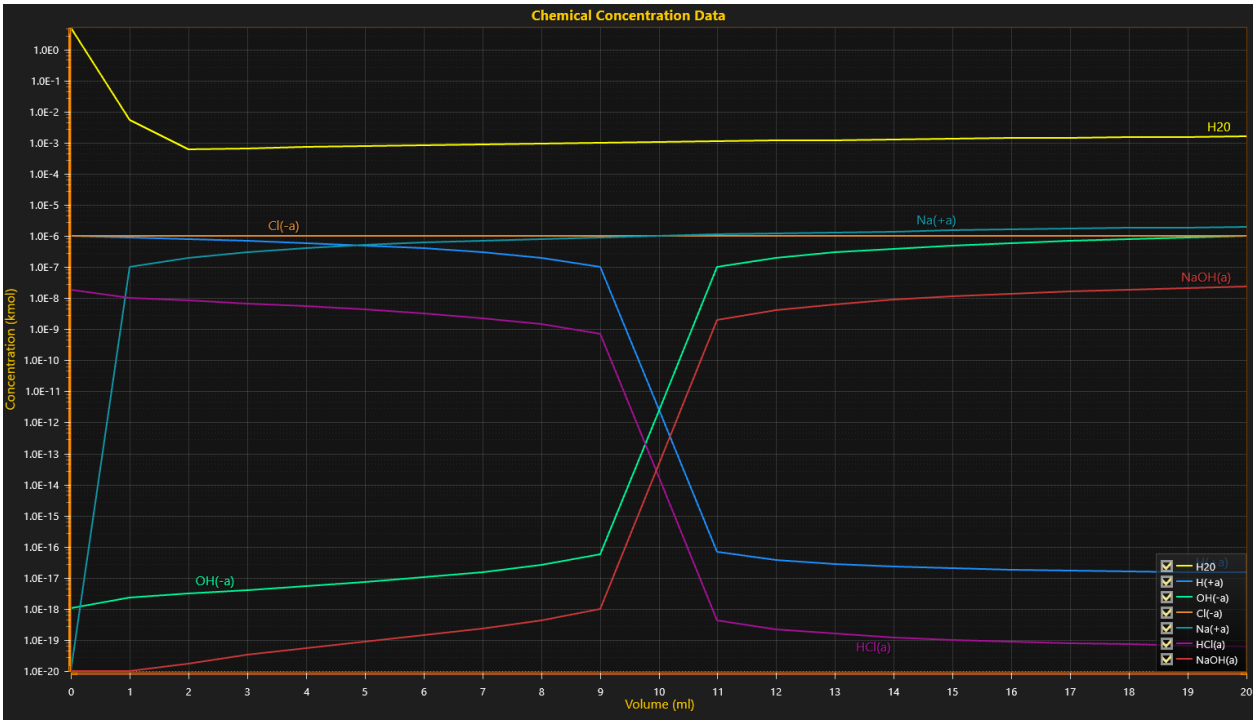


图 6-27. 对数 Y 轴，值接近于零。设置 LogZeroClamp 为 1.0E-20；设置 LogBase 为 10，设置 LogLabelsType 为 Log10Exponential, 以 1.0E 表示法来显示值

6.2.11.2 自然对数

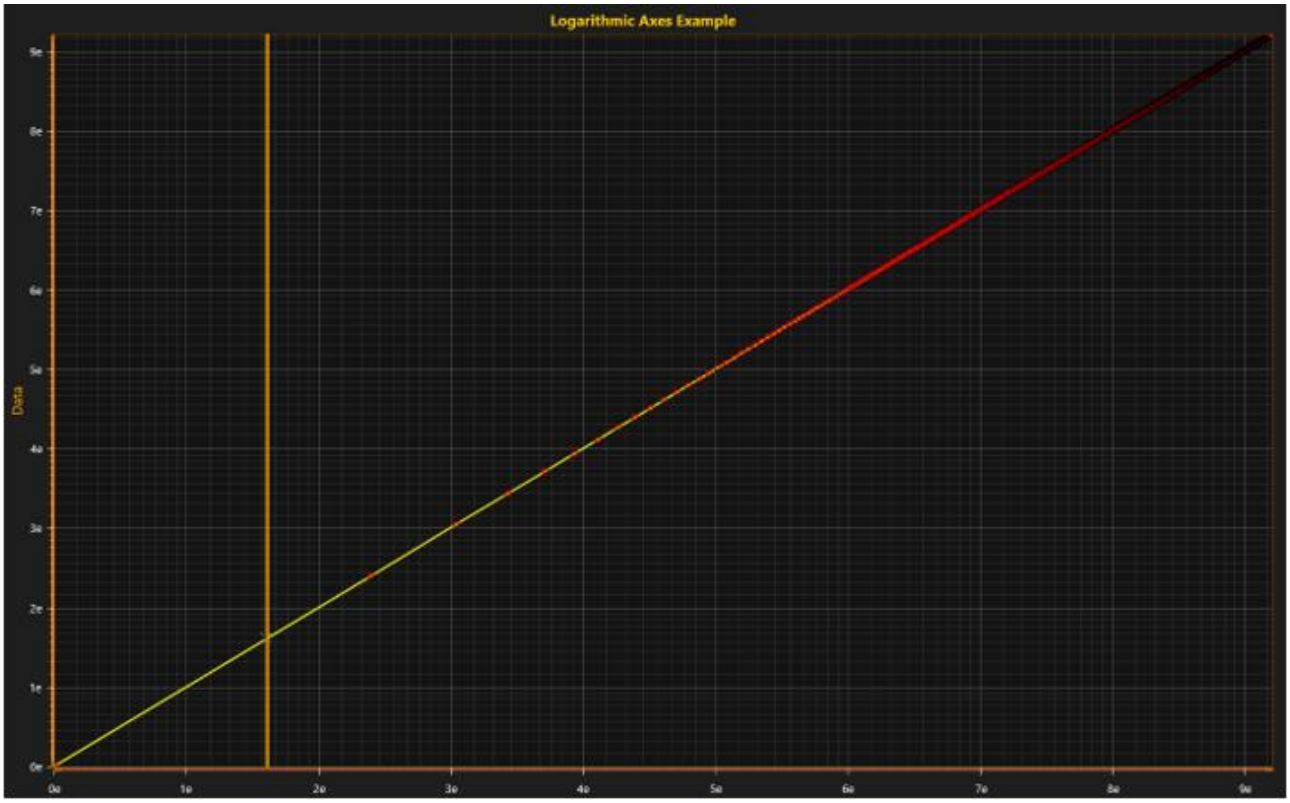


图 6 28. 自然对数视图。设置 LogBase 为 Math.E；设置 LogLabelsType 为 LogE_MultiplesOfNeper.

6.2.12 数轴值与屏幕坐标的转换

轴的方法，可以将轴的值转换为屏幕坐标，以及将屏幕坐标转换为轴值。使用 ***ValueToCoord*** 方法可以把一个轴的值转换为一个屏幕坐标，使用 ***CoordToValue*** 可以把一个屏幕坐标转换为一个轴值。如果首选的是像素点（pixels），不是 DIPs，可以设置 ***UseDIP = False***。

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

在图表有了最终尺寸大小后有 ***ValueToCoord*** 和 ***CoordToValue*** 两个方法可用。例如，订阅 ***AfterRendering*** 事件来确保图表得到充分渲染。

一次要转换多个值或者坐标，可以使用 ***ValuesToCoords*** 和 ***CoordsToValues*** methods。它们以双数组的形式接受/返回轴值（X 轴 ***CoordToValue*** 为整数数组），并以浮点型数组接受/返回屏幕坐标。

```
chart.ViewXY.YAxes[0].CoordsToValues（coordArray, out doubleValueArray, false）;
```

6.2.13 MiniScale

MiniScale 是的 X 和 Y 轴的微型替代。在某些应用程序中，这种比例表示形式更适合用于快速直观地查看数据量，或者在没有空间放下实际的坐标轴时候也适用。通过 ***Visible*** –属性可以激活 ***MiniScale***。***MiniScale*** 是 Y 轴类的一个子属性。不过，X 的比例大小总是绑定到第一条 X 轴（***XAxes[0]***）。修改 X 和 Y 轴的 ***Units.Text*** 属性可以设置单位可见。***MiniScale*** 不能与对数轴一起使用。

```
// 配置 MiniScale
chart.ViewXY.YAxes[0].MiniScale.Visible = true;
chart.ViewXY.YAxes[0].MiniScale.VerticalAlign = AlignmentVertical.Bottom;
chart.ViewXY.YAxes[0].MiniScale.Offset.SetValues（-10, -30）;
chart.ViewXY.YAxes[0].MiniScale.PreferredSize = new SizeDoubleXY（30, 30）;
chart.ViewXY.XAxes[0].Units.Text = "s";
chart.ViewXY.YAxes[0].Units.Text = "µV";
```

图 6-28. 图表右下角是 MiniScale

6.2.14 轴端点标签

规则轴的主要刻度和标签以统一的间隔放置。 因此，轴最小值或最大值可能没有标签，特别是当轴被平移、滚动或对数轴被深度缩放时。通过启用 **EndPointLabelsVisible** 属性，可以强制标签显示在轴的两端。

PreferEndPointLabelsOverNearbyMajorTick 属性控制结束标签或常规主刻度如果它们的位置重叠，则首选标签。 **EndPointMajorTickThreshold** 属性定义了数字在隐藏终点标签之前必须可见的主要刻度线。 默认-1 表示结束点标签将始终可见。 如果对数轴主刻度计数 **<= EndPointMajorTickThreshold**，

然后将显示小刻度旁边的标签。

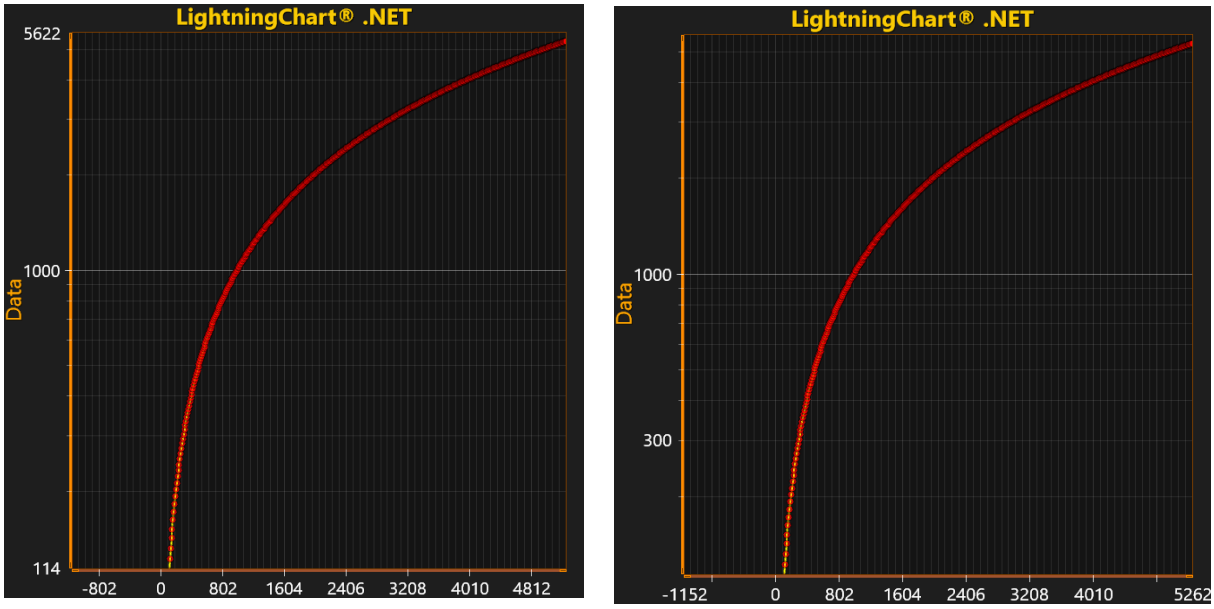


图 6-31。 在左侧图表的 X 轴上 PreferEndPointLabelsOverNearbyMajorTick 被禁用，而在右侧这个属性是启用。 左侧图表的 Y 轴设置为始终显示结束标签（EndPointLabelsVisible=true，EndPointMajorTickThreshold=-1），而右侧图表的 Y 轴配置为除了一个主要刻度外还显示一个次要刻度(EndPointLabelsVisible=true, EndPointMajorTickThreshold=1)

6.3 X 轴

X 轴分度和网格设置与 Y 轴的设置一样。所以之前章节所介绍的属性与功能也同样可以应用到 X 轴。但是，X 轴有几个与实时滚动相关的属性，这是 Y 轴没有的。

6.3.1 实时监控滚动

演示实例： Billion Points; *（温度图）Temperature graph; Thread-fed multi-channel data*

当制定一个实时监控方案时，必须滚动 X 轴来正确显示当前的监控位置，这通常是最新信号点的时间戳。在将新的信号点设置为一个系列之后，将 ***ScrollPosition*** 属性设置为最新的一次时间戳。

```
// 将实时监控滚动条位置设置为最新的 X 值
chart.ViewXY.XAxes[0].ScrollPosition = latestDataPoint.X;
```

LightningChart 中有几个滚动模式可选，使用 ***ScrollMode*** 属性可以进行选择。

```
chart.ViewXY.XAxis[0].ScrollMode = XAxisScrollMode.Scrolling;
```

6.3.1.1 None （不滚动）

此为默认选项。当设置 ***ScrollPosition*** 为 ***None*** 时，即应用不滚动模式。在不适用实时监控的时候通常选择这一项。

6.3.1.2 Stepping （步进）

当采集到的数据到达 X 轴的末端时，带有所有系列数据的轴按步进间隔向左移动。每当到达 X 轴末端时，都执行一次移动。步进值的范围可以通过定义 ***SteppingInterval*** 属性来设置。

```
chart.ViewXY.XAxes[0].SteppingInterval = 3;
```

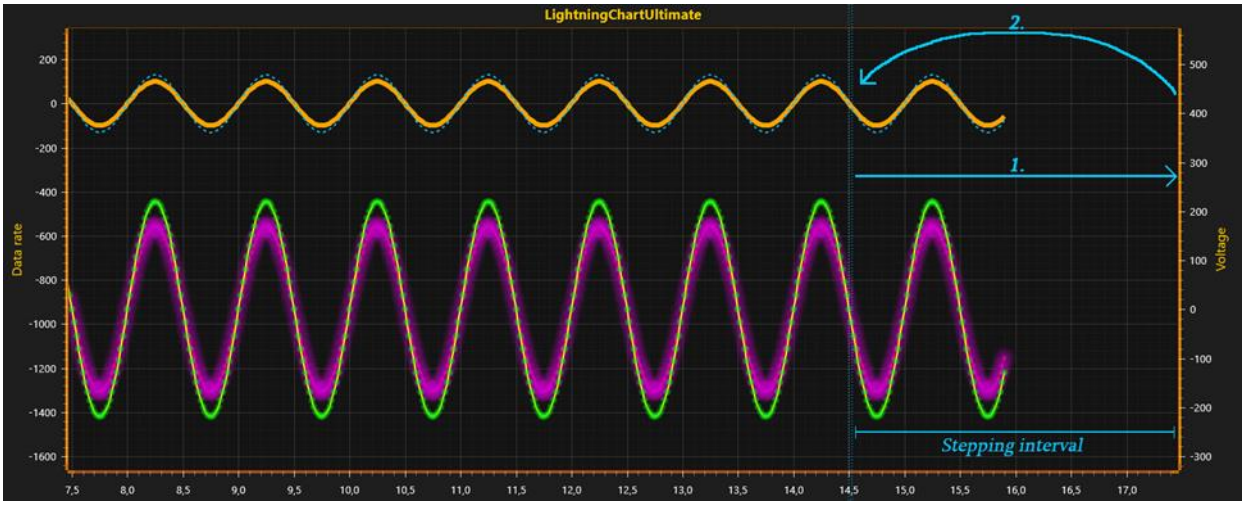


图 6-29. X 轴的滚动模式：步进 stepping

6.3.1.3 Scrolling（滚动）

X 轴开始保持静止状态，待滚动间隔到达设定值后，带有所有系列的 X 轴会连续向左移动。如果要在滚动位置抵达 X 轴末端时再让滚动生效，则设置 ***ScrollingGap*** 为 0。***ScrollingGap*** 属性的定义为图形宽度的百分比。

```
chart.ViewXY.XAxes[0].ScrollingGap = 15;
```

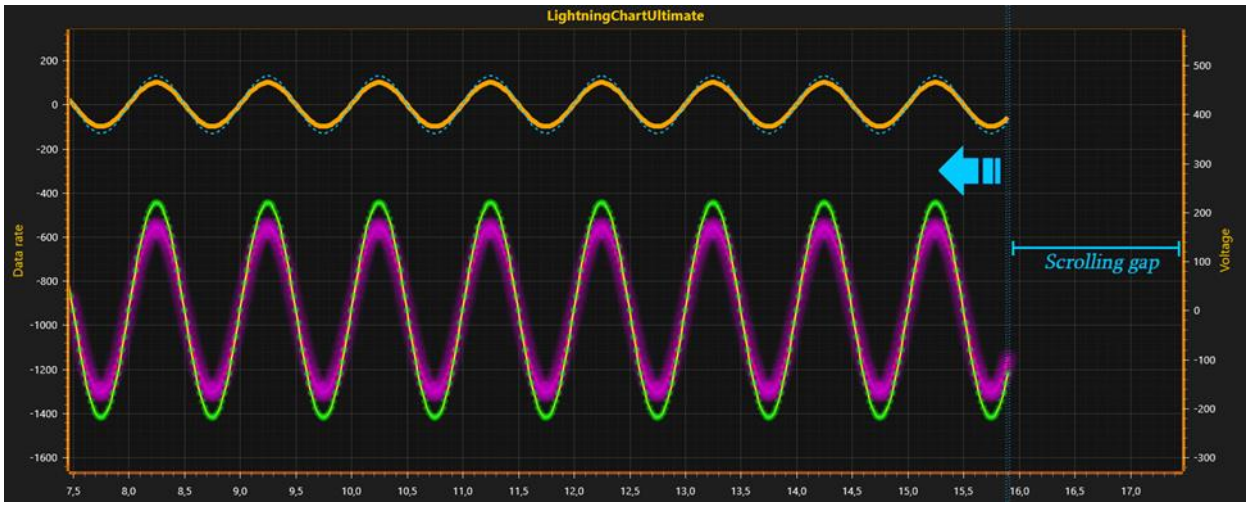


图 6-30. X 轴滚动模式：滚动 scrolling

滚动过程中波形稳定

当使用 ***series.AddPoints***（）、***AddValues***（）或 ***AddSamples***（）-方法后，LightningChart 支持实时信号的增量渲染数据构建。这意味着仅从新的数据部分计算渲染数据，并与现有的渲染数据相结合。

PointLineSeries、***SampleDataSeries***、***AreaSeries*** 和 ***HighLowSeries*** 具有一个特定的属性用于 ***ScrollMode*** = ***Scrolling***，可影响滚动系列的视觉稳定性，保持波形特性。该属性叫做 ***ScrollingStabilizing***.

```
chart.ViewXY.PointLineSeries[0].ScrollingStabilizing = true;
```

在启用 ***ScrollingStabilizing*** 后，浮点坐标四舍五入到最近的整数坐标，结果得到一个视觉上稳定的、无波动的波形。多数情况下，这是一个最佳的方法。不过，在四舍五入坐标时，相位信息可能会有略微的失真。

在启用 ***ScrollingStabilizing*** 后，数据渲染使用浮点坐标，而当 GPU 决定像素坐标时，浮点坐标以轻微波动的波形出现。这带来了更好的视觉特性，特别是在显示正弦数据时，几乎每隔一个像素就会有一次上下转变。

要使用增量渲染数据构造，须按如下方式添加新点

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

```
chart.EndUpdate（）;
```

调用 ***InvalidateData（）***，可以随时刷新渲染数据

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

	性能	稳定性	相位
series.AddPoints（）, ScrollStabilizing *disabled*	完好	受损	良好
series.AddPoints（）, ScrollStabilizing *enabled*	完好	最佳	轻微受损
series.AddPoints（）, InvalidateData（）	受损	受损	完好

表 5-1. 滚动过程中波形稳定

6.3.1.4 扫掠

在扫描模式下的实时监控视图可能是用户友好度最高的。该模式中使用两条 X 轴。第一条轴采集满后，出现一个扫描间隔，然后第二个 X 轴扫过第一个。两个 X 轴都显示各自的值标签。***SweepingGap*** 属性的定义为图形宽度的百分比。

```
chart.ViewXY.XAxes[0].SweepingGap = 5;
```

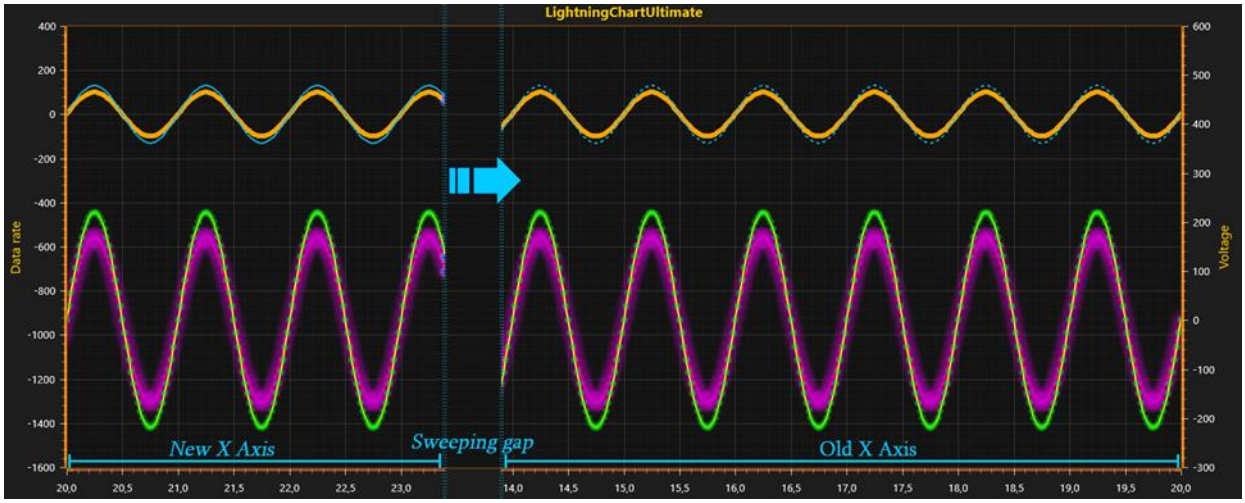


图 6-31. X 轴滚动模式： 扫掠

6.3.1.5 触发（Triggering）

X 轴位置由超过或低于触发位准的系列值决定。使用 ***Triggering*** 属性可设置触发选项。通过启用 ***Triggering.TriggeringActive*** 属性可以设置触发为激活状态。

必须将一个系列设置为触发系列。可接受的触发系列类型有 ***PointLineSeries*** 和 ***SampleDataSeries***。使用 ***Triggering.TriggerLevel*** 可设置触发 Y level 。使用 ***Triggering.TriggeringXPosition*** 可按图形宽度的百分比的方式，确定水平绘制位准触发点的位置。

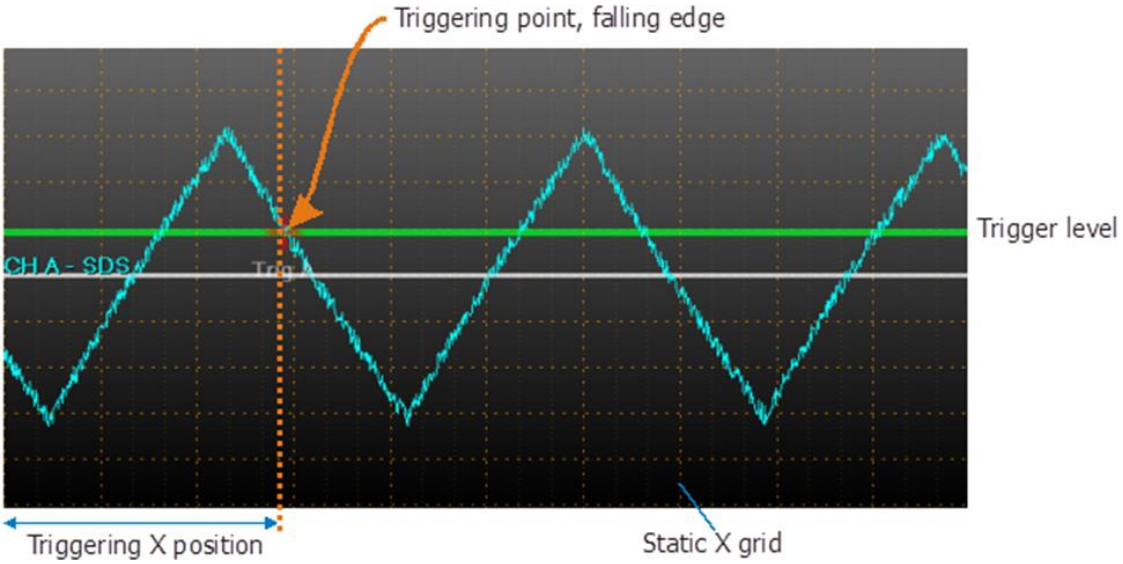


图 6-32. X 轴滚动模式：由静态 X 网格触发

当使用触发的 X 轴滚动位置时，通常不适合显示带有值和网格的常规 X 轴，因为该位置根据输入的系列数据会从一个位置跳到另一个位置。

- 方法 1：使用静态 X 网格。通过设置 ***XAxis.Visible = false***（或者 ***LabelsVisible*** = false，***MajorGrid.Visible*** = false 并且 ***MinorGrid.Visible*** = false），可以隐藏常规的 X 轴对象。然后，设置 ***Triggering.StaticMajorXGridOptions*** 和 ***Triggering.StaticMinorXGridOptions*** 来显示静态 X 网格。

- 方法 2：使用首选刻度创建另一条 X 轴，并设置为 ViewXY 集合。不要为该系列分配第二条 X 轴。

对于刻度指示，使用 Y 轴的 **MiniScale** 或定义一个 **Annotation** 对象（参阅第 6.26 章节）来显示值域范围，例如“200 ms/div”。

6.3.2 刻度中断（Scale breaks）

演示示例： Scale breaks; Stock course with previous close

从第 8 版本开始，X 轴支持 ***ScaleBreaks***。采用 ***ScaleBreaks*** 可以将特定的 X 值域范围排除在外，例如不活跃的交易时间/日期或机器停产时间。所有分配到指定 X 轴的系列都被剪切，包括轴和标签。

可以使用 ***ScaleBreaks*** 的情况是有限制的：必须设置 ***ScrollMode*** 为 ‘***None’***，同时设置 ***ScaleType*** 为 ‘***Linear***’。

在 X 轴的 ***ScaleBreaks*** 集合中插入 ***ScaleBreak*** 对象。

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

图 6-33. ScaleBreak 属性

用 ***Begin*** 和 ***End*** 可以指定中断的范围，二者是以轴值的形式设置，不是 DateTimes 。如果要采用 DateTimes 可以使用 axis.***DateTimeToAxisValue*** 来进行转换。

间隔宽度可以用 ***Gap*** 进行调整，如果不显示间隔也可以设为 0。用 ***Style*** 可以对间隔外观进行配置。

- 设置 ***Style*** = '***Fill***'，用 ***Fill*** 属性调整填充样式。
- 设置 ***Style*** = '***DiagonalLineUp***' 或 '***DiagonalLineDown***'，用 ***DiagonalLineSpacing*** 和 ***LineStyle*** 属性调整外观。

设置 ***Enabled*** = ***False***，可取消中断。

PointLineSeries、 ***AreaSeries*** 和 ***HighLowSeries*** 具有 ***ContinuousOverScaleBreak*** 属性。启动该属性后，可以在间距之间绘制一条连接线。

图 6-34. 原始交易数据，周一至周五，上午 10 点至下午 6 点。没有应用 ScaleBreaks。大部分时间范围内并没有数据，这是因为证券交易所已经关闭，难以看到必要的信息。PointLineSeries 从收市值开始跳变。

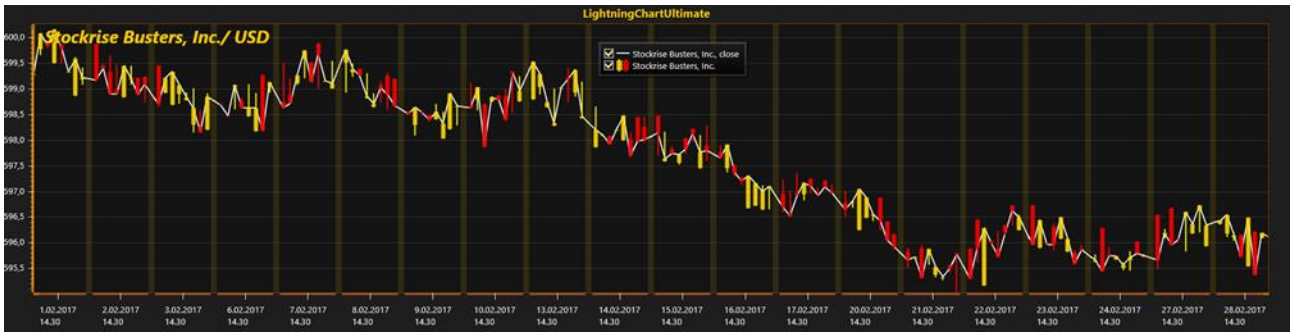


图 6-35. 应用 ScaleBreaks 可消除无活动交易时间。这样就有更多屏幕空间能显示出重要的数据。Style = Fill, Gap = 10. PointLineSeries 从收市值开始跳变, PointLineSeries.ContinuousOverScaleBreak = True.

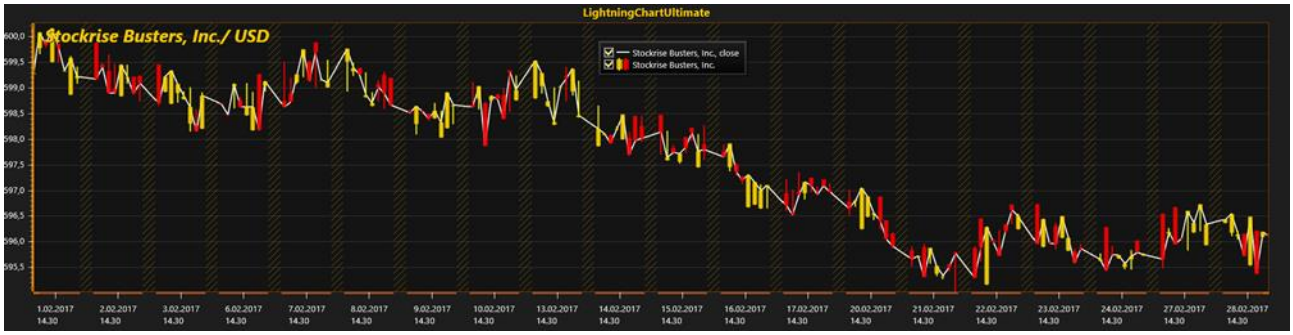


图 6-36. 在无活动交易时间应用 ScaleBreaks； Style = DiagonalLinesUp, Gap = 20. PointLineSeries.ContinuousOverScaleBreak = True.

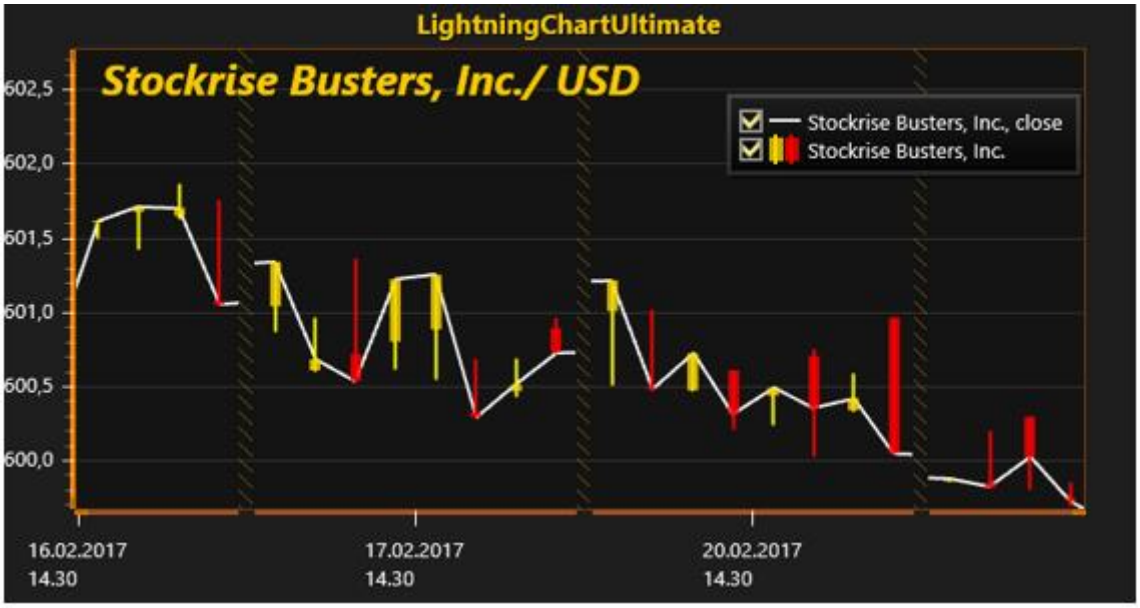


图 6-37. PointLineSeries.ContinuousOverScaleBreak = False. 在间隔间，前一点到下一点的线不相连。相反，这些线继续保持原来的方向，就像没有定义尺度中断一样。

6.4 图边距

图边距是图形区域周围的空白处。除了注释、图例框和图表标题之外，视图的所有内容都显示在图边距内。

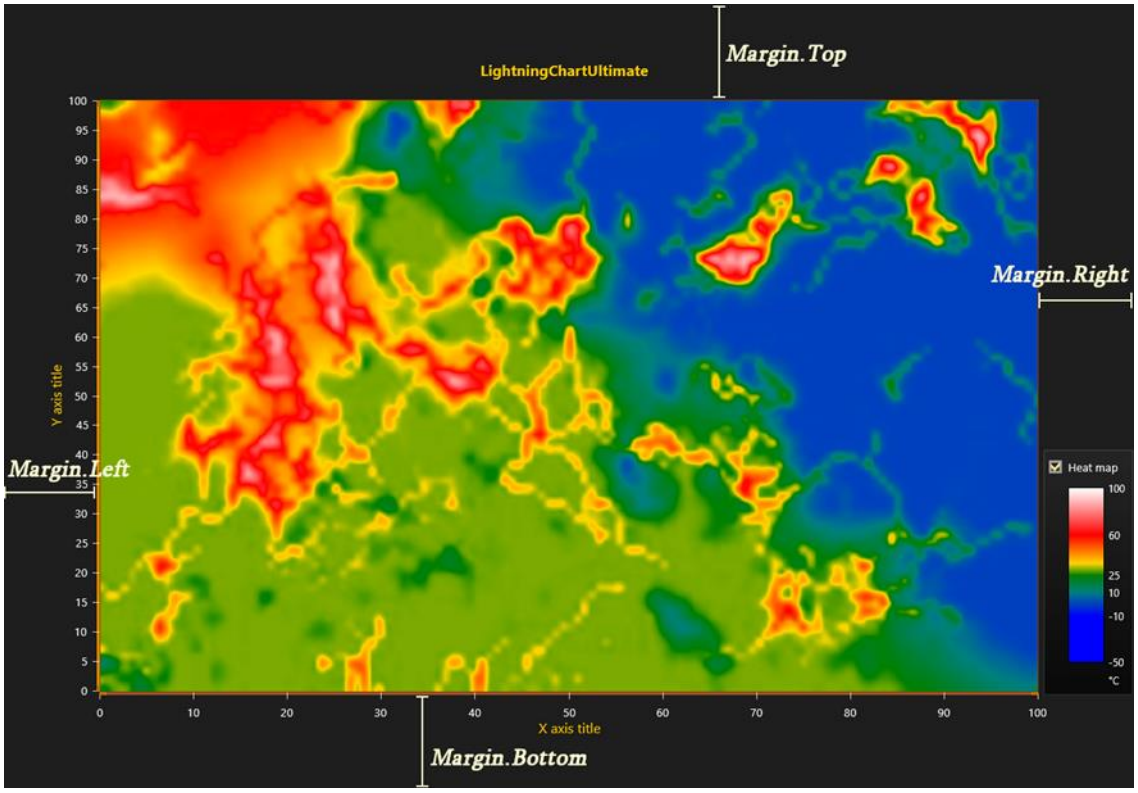


图 6-38. 图表区域周围为图边距。图边距内填置具体内容。图边距上可以放置图表标题和图例框。

当 ***AutoAdjustMargins*** 设为 ***enabled*** 后，会调整图表尺寸，腾出足够的空间放置所有的轴和图表标题。设为 ***disabled*** 后，应用 ***ViewXY.Margins*** 属性可以手动设定图边距。

默认情况下，在图表区域的图边框位置绘制可以自定义绘制一个矩形边框 ***Border***。通过设置 ***Border.Visible = False*** 可以关闭该边框。***Border*** 的颜色也可以通过 ***Color*** 属性更改。另外，通过设置 ***RenderBehindSeries*** 为 ***True***，也可以在系列后渲染 ***Border***。

在运行期间，通过调用 ***ViewXY.GetMarginsRect*** 方法 method，可以恢复以像素为单位的图边距矩形，适用于自动和手动图边距。当需要执行基于屏幕坐标的计算或对象放置时，这非常有用。

ViewXY.MarginsChanged 事件可以设置为在图边距矩形变更时（例如调整其大小）触发。

6.5 ViewXY 系列，通用

ViewXY 的系列以不同的方式和格式实现数据可视化。所有系列都连接到轴值范围；而且，系列必须连接到一条 Y 轴。通过系列的 ***AssignXAxisIndex*** 和 ***AssignYAxisIndex*** 属性可以分配 X 和 Y 轴。通过代码，使用系列构造函数参数交替分配 X 轴和 Y 轴。

6.5.1 线条样式

演示示例：*Minimal logarithmic value*

每个 XY 系列都有标题（默认隐藏），其样式和位置可以通过修改属性（系列.Title.propertyName）。启用 **ViewXY.TitlesAutoPlacement.Enabled** 通知图表该系列的标题位置应自动计算以避免多个标题重叠.

通过启用 series.Title.LockAutoPosition 可以将标题位置锁定在当前 X-Y 值财产。 使用 **ViewXY.TitlesAutoPlacement.Reset()** 方法重置自动定位。

6.6 PointLineSeries

演示示例： 点线图（Point line）；温度图（Temperature graph）；线、调色板着色（Line, palette coloring）

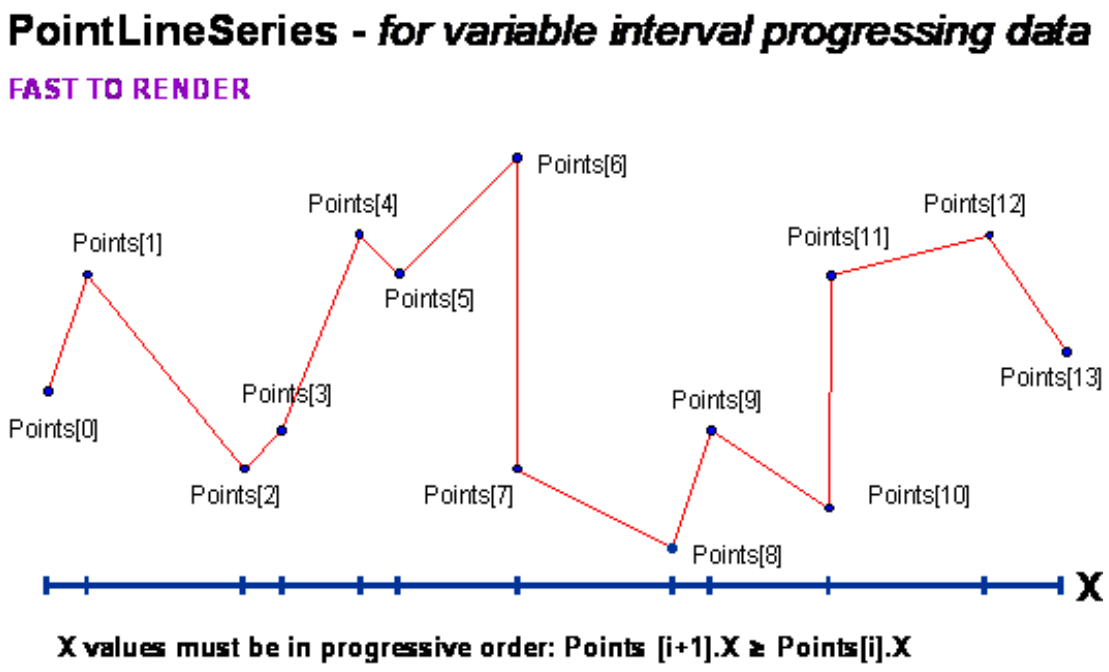


图 6-39. PointLineSeries 概览

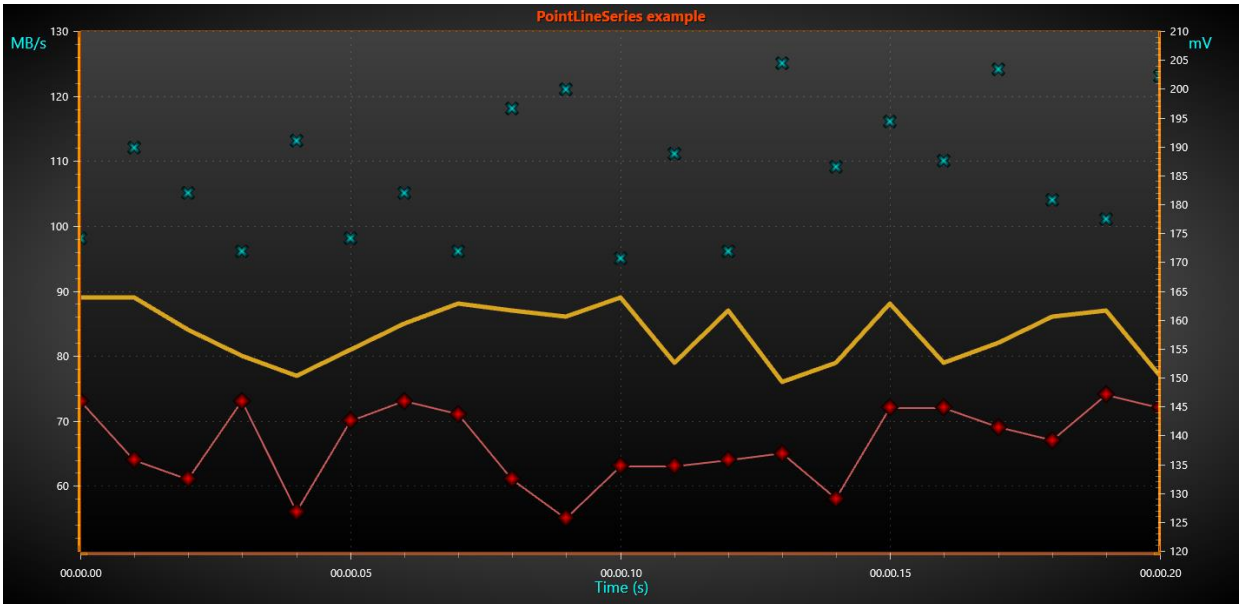


图 6-40. 三种不同的 PointLineSeries.

一个 ***PointLineSeries*** 能够展现一个基本的线、点（散点图）或者二者合一的点线图。通过向 ***PointLineSeries*** 列表添加 ***PointLineSeries*** 对象，可以向图表添加系列。

```
chart.ViewXY.PointLineSeries.Add（series）; // 向图表添加系列
```

6.6.1 线条样式

所有线系列都可以呈现一条线（在 2 个或更多点之间）。 该系列具有控制颜色的属性 LineStyle 类下的线条和宽度。 如果线条不可见，请设置 **LineVisible = false**。 除了这些属性之外，还可以修改线型。 可用选项包括实心、点、破折号、DashDot 和 SmallDot。 可以为 **SampleDataSeries**、**PointLineSeries**、**FreeformPointLineSeries**、**AreaSeries**、**HighLowSeries** 和 **LineCollection**。 对于这些系列 PatternScale 属性可用于修改每个破折号或点的长度。

仅使用 **SampleDataBlockSeries**、**LiteLineSeries**、**LiteFreeformLineSeries 和 DigitalLineSeries**

可以画线。

6.6.2 点样式

设置 ***PointsVisible = true*** 可以显示点。通过设置 ***PointStyle*** 属性可以变更点的样式。通过 ***PointStyle.Shape*** 从众多预定义的样式中选择点的外形。其中一种是 ***Bitmap***，即可以在点位置上绘制任意的位图图像。使用 ***BitmapImage*** 属性可以定义位图图像。用 ***BitmapAlphaLevel*** 属性可以变更位图的透明度。通过变更 ***BitmapImageTintColor***，可以调整位图的色调为白色之外的其他银色调。当使用预定义的点样式时，如 **Circle**（圆形）、**Triangle**（三角形）、**Cross**（十字）等，可以定义绘图颜色和填充样式。注意，并不是所有的颜色或填充适用于所有形状样式。点的宽度和高度可以设置，点也可以旋转。

6.6.3 单独为点配色

演示示例： Point line, individually colored points, Scatter points, individually colored

从第 v.7.2 版本开始， ***PointLineSeries***、***FreeformPointLineSeries***、***AreaSeries*** 和 ***HighLowSeries*** 在数据点结构中具有 ***PointColor*** 字段。

要开启给点单独着色，要在 ***IndividualPointColoring*** 设置 ***Color1***、***Color2***、***Color3*** 或 ***BorderColor***。 要禁用单独给点着色，则设置 ***IndividualPointColoring*** = ***Off***。颜色设置与 ***PointStyle*** 属性中的颜色相当。

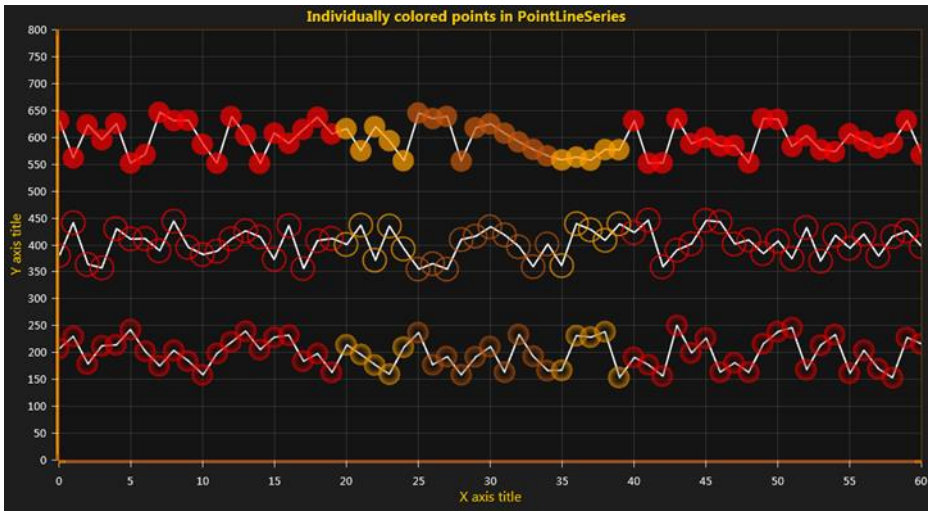


图 6-41. 顶部，设置 IndividualPointColoring = Color1 （实心着色点）；中间，设置 IndividualPointColoring = BorderColor；底部，IndividualPointColoring = Color2 （采用 Color1 = transparent 设置渐变着色）。

6.6.4 添加点

系列点必须在代码中添加。用 ***AddPoints（SeriesPoint[], bool invalidate）***方法在现有的点的末端添加点。

```
chart.ViewXY.PointLineSeries[0].AddPoints（pointsArray）; //在末端添加点
```

若要一次性设置全部系列数据，并覆盖旧的点，可以直接分配新点数组：

```
chart.ViewXY.PointLineSeries[0].Points = pointsArray; //分配点数组
```

注意！PointLineSeries 点的 X 值必须按升序排列。如果一定要对其另外排序，可用 FreeformPointLineSeries 替代。

例如，定义 Points[0].X = 0, Points[1].X = 5, Points[2].X = 5, Points[3].X = 6 是有效的。

但 Points[0].X = 2, Points[1].X =1, Points[2].X = 6, Points[3].X =7 对于 ***PointLineSeries*** 来说不是一个有效的值数组。

6.6.5 添加点的其他方法

X 和 Y 值数组中也可以添加点，这在许多应用程序中会更为方便。

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

若要一次性设置全部系列数据，并覆盖旧的点，可以直接分配 X 和 Y 值数组：

（适用于 WinForms 和 WPF Non-bindable APIs）

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

6.7 LiteLineSeries

LiteLineSeries 是 PointLineSeries 的一个版本，针对性能进行了优化。它与 PointLineSeries 类似，但配置选项较少。 LiteLineSeries 仅绘制在数据点之间线条，但不是点本身。 此外，它只有颜色和宽度属性来调整系列外观。 LiteLineSeries 期望数据点按递增顺序排列。

使用 AddPoints() 方法将数据点数组添加到系列中。 这些数组应该是 double[,] 类型。其中第一个值是数据点索引，而第二个值同时具有 X 值和 Y 值，可以调用 ActualPointCount() 来找出点的总数。

```
// Adding a LiteLineSeries with some random data points.
LiteLineSeries lls = new LiteLineSeries(_chart.ViewXY,
_chart.ViewXY.XAxes[0], _chart.ViewXY.YAxes[0]);
lls.Width = 2;
lls.Color = Colors.Lime;
double[,] values = new double[21, 2];
for (int i = 0; i < 21; i++)
{
values[i, 0] = i;
values[i, 1] = rand.NextDouble() * 100;
}
lls.AddPoints(values, false);
_chart.ViewXY.LiteLineSeries.Add(lls);
```

6.8 SampleDataSeries

演示示例：演示示例： Billion points; Thread-fed multi-channel data; Signal reader

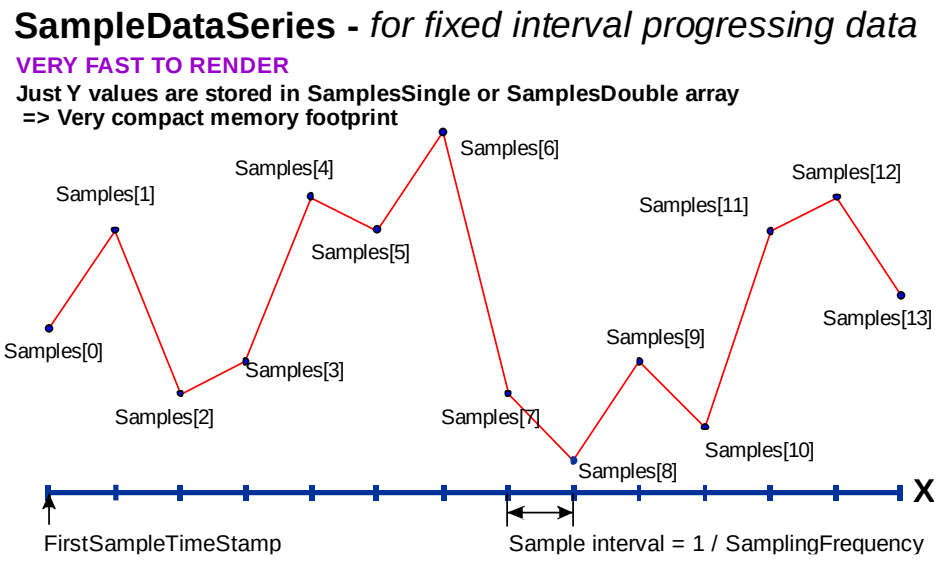


图 6-42. SampleDataSeries 概览

通过向 ***SampleDataSeries*** 列表中添加 ***SampleDataSeries*** 对象，可以向图表中添加系列。

chart.ViewXY.SampleDataSeries.Add（sampleDataSeries）; //向图表中添加一个 SampleDataSeries

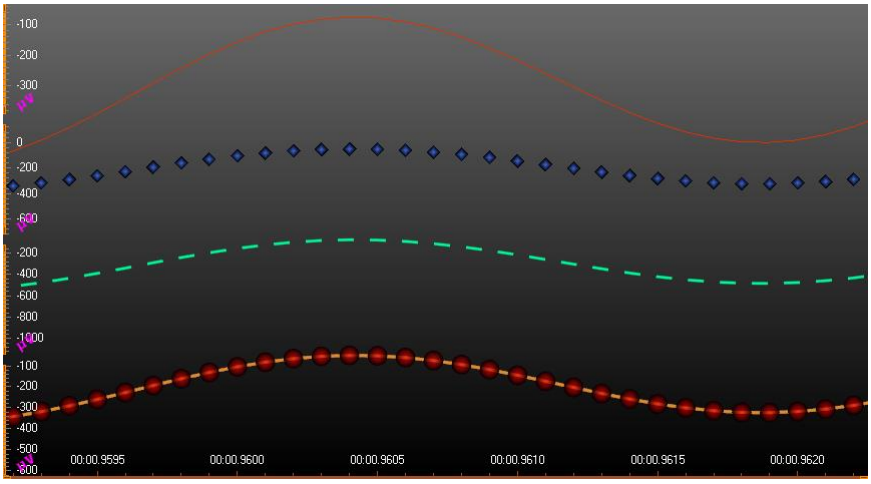


图 6-43. 一些样本数据系列.

SampleDataSeries 是一种线系列，用以表示采样的信号数据（离散信号的数据）。通常用于实时 DSP 应用程序。外观上看，与 ***PointLineSeries*** 类似，因此，所有的线和点格式的选项都适用。由于 ***SampleDataSeries*** 具有固定的采样间隔，所以不需要保留内存来存储点的 X 值。

注意！*SampleDataSeries* 不对给定的数据重新采样或降采样。所有给定的数据保存在 ***SamplesSingle*** 或 ***SamplesDouble*** 数组中。LightningChart 不会降低数据的特性或丢损数据的峰值或准确性。

6.8.1 Y 精度

SampleDataSeries 支持单精度与双精度样本 Y 值。当要尽可能低的保留内存时，建议采用单精度值。用 ***SampleFormat*** 属性选择采样格式 。

用系列 的 ***SamplingFrequency*** （1 / 采样间隔）来设置固定的采样间隔。要在采样开始设置 X 值（时间戳）需设置 ***FirstSampleTimeStamp*** 属性。

6.8.2 添加点

样本必须在代码中添加。用 ***AddSamples*** 方法在现有样本末端增添样本。

chart.ViewXY.SampleDataSeries[0].AddSamples（samplesArray,false）; //在末端增加样本

若要一次性设置全部系列数据，并覆盖旧的样本，可以直接分配新样本数组：

如果设置 ***SampleFormat*** 为 ***SingleFloat***

chart.ViewXY.SampleDataSeries[0].SamplesSingle = samplesSingleArray;
或者如果设置 ***SampleFormat*** 为 ***DoubleFloat***

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

6.9 SampleDataBlockSeries

SampleDataBlockSeries 是 SampleDataSeries 的一个版本，针对实时应用程序进行了完全优化。 它以最少的 CPU 和内存消耗提供了最佳的性能，从而允许同时渲染极大量的数据点。 就像该系列的名称一样，数据在内部作为块进行管理，而这些模块又分别由内存管理。 这消除了对非常大的连续线性存储器的需求。 SampleDataBlockSeries 用于实时医疗监控应用程序（例如 ECG / EKG，EEG，工业监控应用程序，遥测和波形振动监控）的最佳 series 类型。

SampleDataBlockSeries 的工作原理几乎与 SampleDataSeries 类似。 同样地，它要求所按渐进的顺序添加数据，并应具有固定的数据间隔。 SamplingFrequency（1 /采样间隔）可用于设置固定的采样间隔。 若要设置样本开始处的 X 值（time stamp），请设置 FirstSampleTimeStamp 属性。 但是，与其他 line series 相比，从外观上看，SampleDataBlockSeries 的格式化选项更少。 颜色和宽度属性可用于分别更改线条的颜色和宽度。 此外，SampleDataBlockSeries 仅显示直线，而不显示单个点。

可以使用 AddSamples 方法在代码中添加新样本。 与 SampleDataSeries 不同，SampleDataBlockSeries 仅接受浮点值。 PointCount 属性可用于获取系列中的当前样本数。

```
// Add samples to the end.
sampleDataBlockSeries.AddSamples(samplesArray, false);

// Get the total number of samples.
int samplesCount =_chart.ViewXY.SampleDataBlockSeries[0].PointCount;
```

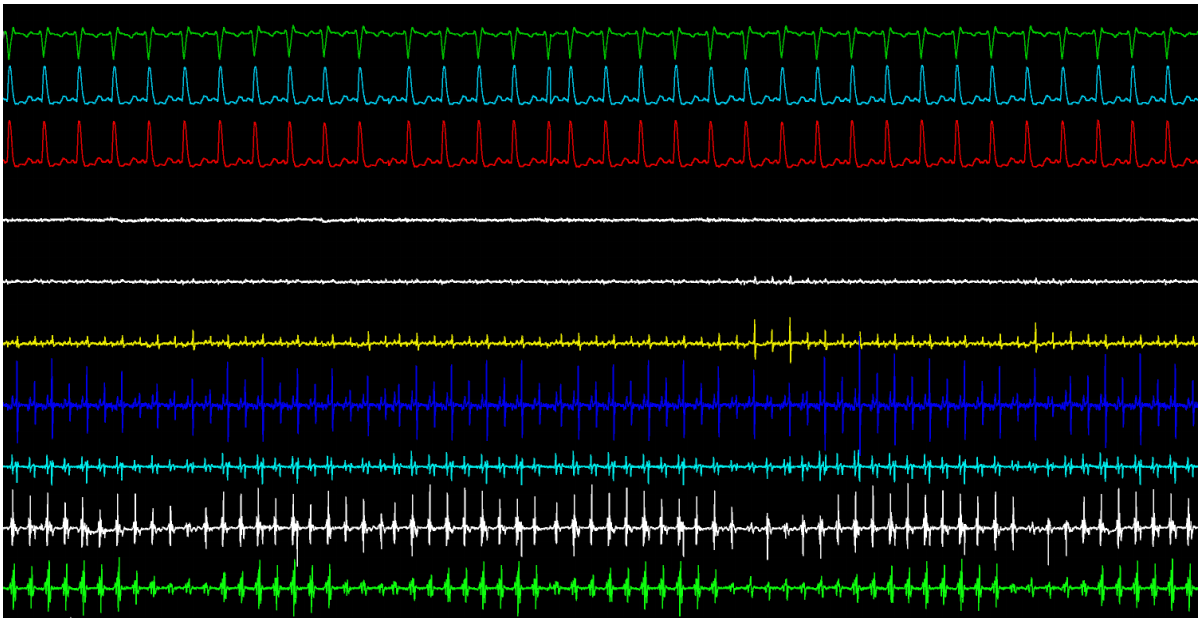


Figure 6-44. 实时应用程序中的几个 SampleDataBlockSeries.

6. 10 DigitalLineSeries

DigitalLineSeries 是一种特定类型的线系列，它显示在两个 Y 值之间交替的线。例如 0 和 1。它针对性能进行了全面优化，并且使用所有系列类型中最少的的内存量。与许多其他系列相比，DigitalLineSeries 的配置选项更少，因为它绘制只有数据点之间的线，而不是点本身。此外，它只有颜色和宽度属性来调整系列外观。

DigitalLineSeries 数据点始终以具有固定间隔的渐进顺序排列。 FirstSampleTimeStamp 属性设置第一个数据点的 X 值，而 SamplingFrequency 控制之间的间隔点。 使用 DigitalHigh 和 DigitalLow 设置线条交替的 Y 值。 数据点通过添加 AddBits() 方法成为 uint[] 类型的数组。 数组中的每个值都转换为相应的二进制值，因此代表 32 个数据点。 BitCount 属性可用于检查增加的点的总数。

```
// Adding a DigitalLineSeries.
DigitalLineSeries dls = new DigitalLineSeries(_chart.ViewXY,
_chart.ViewXY.XAxes[0], _chart.ViewXY.YAxes[0]);
dls.Color = Colors.Yellow;
dls.Width = 2;
dls.FirstSampleTimeStamp = 0;
dls.DigitalLow = 0;
dls.DigitalHigh = 1;
dls.SamplingFrequency = 32;
uint[] data = new uint[] { 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF,
0x00000000, 0xa54df810, 0x00000000, 0xFFFFFFFF };
dls.AddBits(data, false);
_chart.ViewXY.DigitalLineSeries.Add(dls);
```

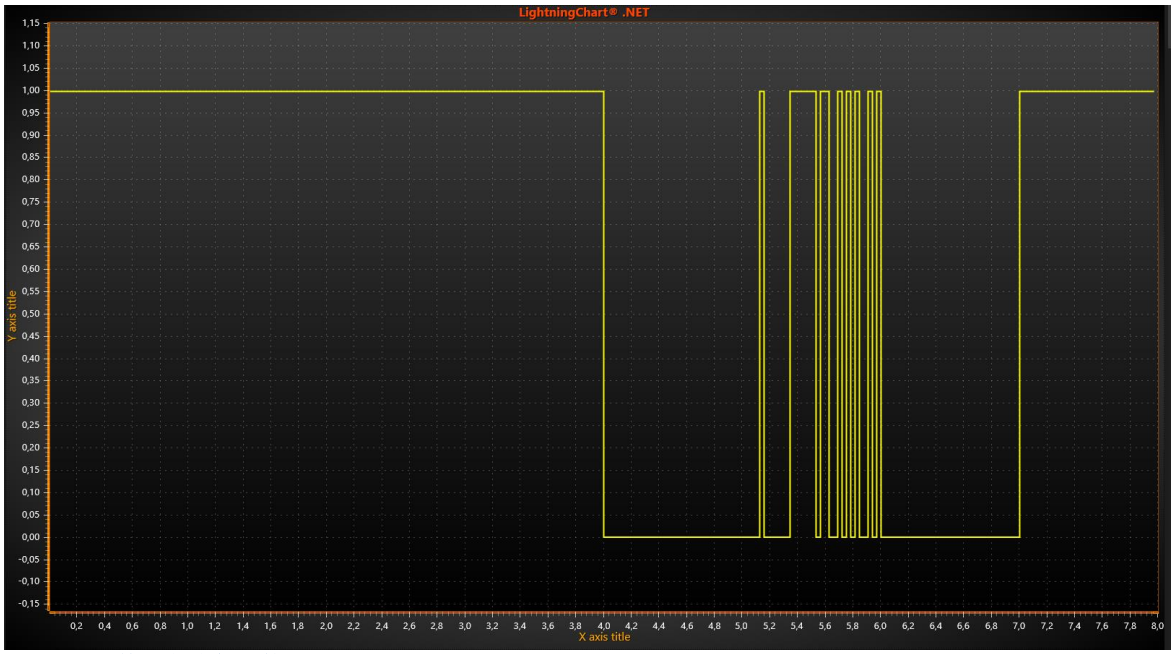


Figure 6-45. 基于上述代码的 DigitalLineSeries

6.11 FreeformPointLineSeries

演示示例：散点（Scatter points）；地图路线（Map route）：Value tracking with markers; Curve node editing

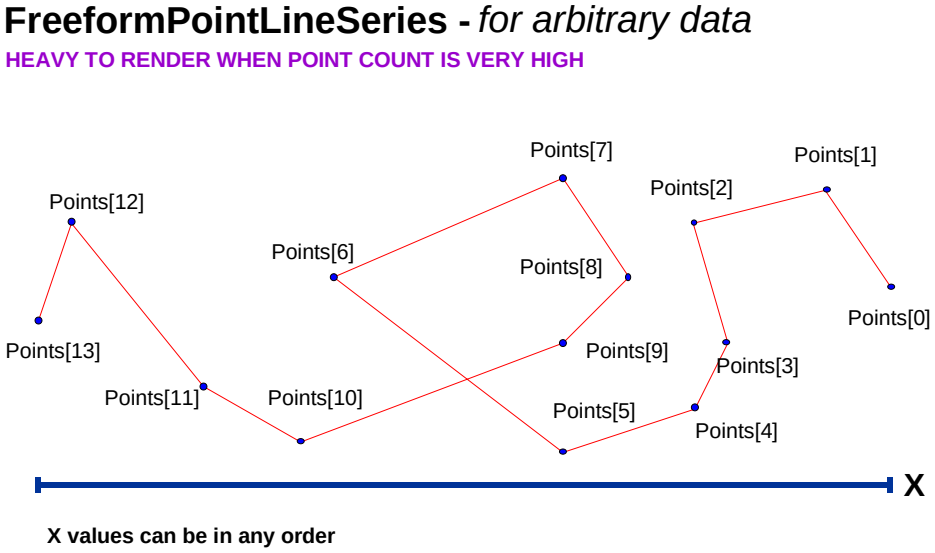


图 6-46. FreeformPointLineSeries 概览

FreeformPointLineSeries 能够展现一个基本的线、点（散点图）或者二者合一的点线图。***FreeformPointLineSerie*** 可以从前一个点向任意方向绘制线点。适用于 ***PointLineSeries*** 中所有的线和点格式化选项。通过向 ***FreeformPointLineSeries*** 列表中添加 ***FreeformPointLineSeries*** 对象，可以向图表中添加系列。

```
chart.ViewXY.FreeformPointLineSeries.Add（freeformPointLineSeries）; //向图表中添加一个FreeformPointLineSeries
```

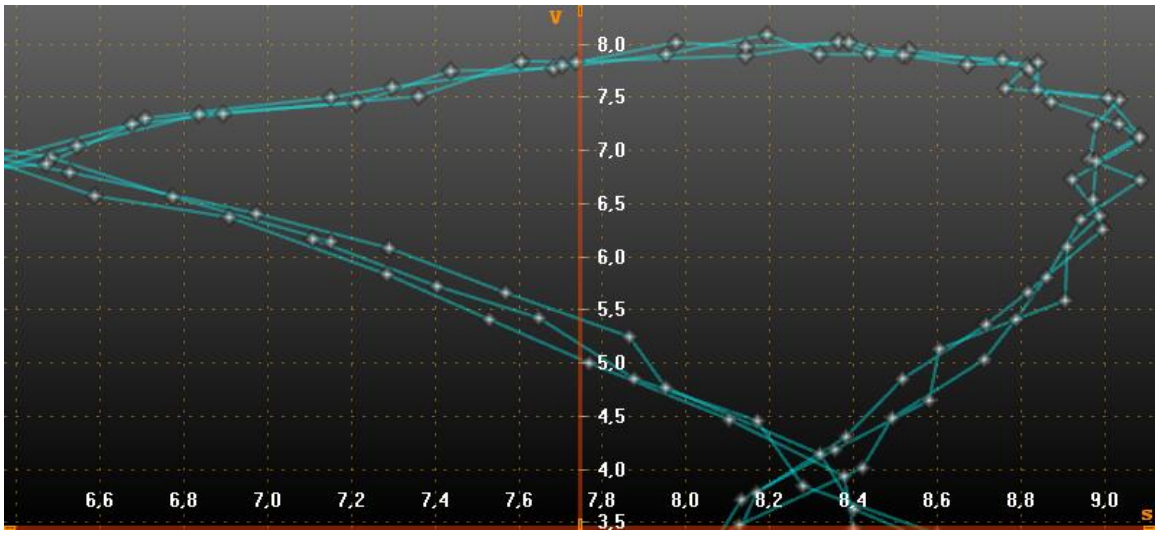


图 6-47. 一个自由式点线系列图

即使启用了 **DropOldSeriesData**，并且点已滚动出当前视图，自由式点线系列的线点也不会自动销毁。要在实时监控解决方案中自动销毁旧的系列点，可以使用点计数限制器 point count limiter。设置 ***PointCountLimitEnabled = true***，并对 ***PointCountLimit*** 属性设置限制。如果限制器开启，在达到点计数限制之后，***Points*** 数组表现为一个环形缓冲区。**Points** 数组中最早的点总是可以通过从 ***OldestPointIndex*** 检索值来找到。如果需要读取现有的数据点计数有限缓冲区，可使用以下方法:

- 如果 ***OldestPointIndex*** 为 0, 从 ***Points*[0]** 开始读取，直到 ***Points*[*PointCount*-1]**.
- 如果 ***OldestPointIndex*** > 0, 首先从 ***Points*[*OldestPointIndex*]** 读取到 ***Points*[*PointCountLimit*-1]**。然后，从***Points*[0]** 读取到***Points*[*OldestPointIndex*-1]**。

要直接检索最后的系列点，可以调用 ***GetLastPoint*（）**方法。

.....

6.12 LiteFreeformLineSeries

LiteFreeformLineSeries 是 FreeformPointLineSeries 的轻量级版本，经过优化，速度更快。但是，与常规系列相比，它的配置选项更少。LiteFreeformLineSeries 仅绘制数据点之间的线，但不绘制点本身。因此不适用于散点图。 此外，它只有颜色和宽度属性可以调整系列外观。 iteFreeformLineSeries 允许自由放置数据点。 换句话说，这些点不必按渐进顺序。

使用 AddPoints() 方法将数据点数组添加到系列中。 这些数组应该是 double[,] 类型，其中第一个值是数据点索引，而第二个值同时具有 X 值和 Y 值 可以调用 ActualPointCount() 来找出点的总数。

```
// Adding a LiteFreeformLineSeries with random data points.
LiteFreeformLineSeries flls = new LiteFreeformLineSeries(_chart.ViewXY,
_chart.ViewXY.XAxes[0], _chart.ViewXY.YAxes[0]);
flls.Color = Colors.Red;
flls.Width = 3;
double[,] values = new double[21, 2];
for (int i = 0; i < 21; i++)
{
values[i, 0] = rand.NextDouble() * 20;
values[i, 1] = rand.NextDouble() * 100;
}
flls.AddPoints(values, false);
_chart.ViewXY.LiteFreeformLineSeries.Add(flls);
```

6.13 应该使用哪个系列

选择线系列取决于三个主要问题：应该显示多少数据点同时，数据点的性质是什么，或者说是按固定顺序排列的点间隔，以及线条在视觉上的外观。

数据点数量/性能

在性能方面，使用正确的系列类型非常重要。 例如，在实时应用程序中使用 **FreeformPointLineSeris** 比 ***SampleDataBlockSeries*** 要重。

- 如果数据点总数较低（< 1000），则系列类型没有明显在性能上的影响。

- 在实时应用中，根据数据点的实际情况选择使用最快线系列。

该系列从渲染最快到最重的性能顺序是：

- DigitalLineSeries

- SampleDataSeries / SampleDataBlockSeries

- PointLineSeries / LiteLineSeries

- FreeformPointLineSeries / LiteFreeformLineSeries

数据点的性质

数据点的性质直接影响系列类型的选择

- 如果点的 X 值不是递增顺序，请使用 FreeformPointLineSeries 或 LiteFreeformLineSeries。

- 如果 X 值是渐进的，但点之间的间隔不同，请使用 PointLineSeries 或 LiteLineSeries。

- 如果 X 值是按固定数据点间隔递增的，请使用 SampleDataSeries、SampleDataBlockSeries 或 DigitalLineSeries。

- 如果数据的 Y 值在两个值之间交替，请考虑使用 DigitalLineSeries

视觉外观

在大多数情况下，同一系列类型有两个版本，例如 PointLineSeries 和 LiteLineSeries。 选择其中一个是在性能和能够完全修改该系列的视觉外观之间进行权衡。 一般来说，所有“lite”系列类型渲染速度更快，使用内存更少，但配置选项少。

- 如果应该渲染数据点本身，而不仅仅是线条，请使用常规系列，因为“lite”系列仅绘制线条。

- 如果修改颜色和线条的宽度就足够了，请使用“lite”系列。

6.14 线系列的高级线着色

演示示例： Line, palette coloring; Line, event-based coloring by indices; Line, event-based coloring

可以根据数据值或其他外部逻辑更改线条颜色。

6.14.1 基于 Y 值的线条着色以及用值域调色板填充

通过启用 ***SampleDataSeries***、***PointLineSeries*** 或 ***FreeformPointLineSeries*** 的 ***UsePalette*** 属性，可以通过 ***ValueRangePalette*** 属性来应用线条着色。***ValueRangePalette*** 包含 Y 值和颜色对。通过 ***ValueRangePalette.Type*** 来设置 ***Gradient*** 或 ***Uniform*** 多阶调色板。

也可以为 Y 轴设置调色板颜色。开启 Y 轴的 ***UsePalette*** 属性，并在 ***PaletteSeries*** 属性中分配优先的系列。

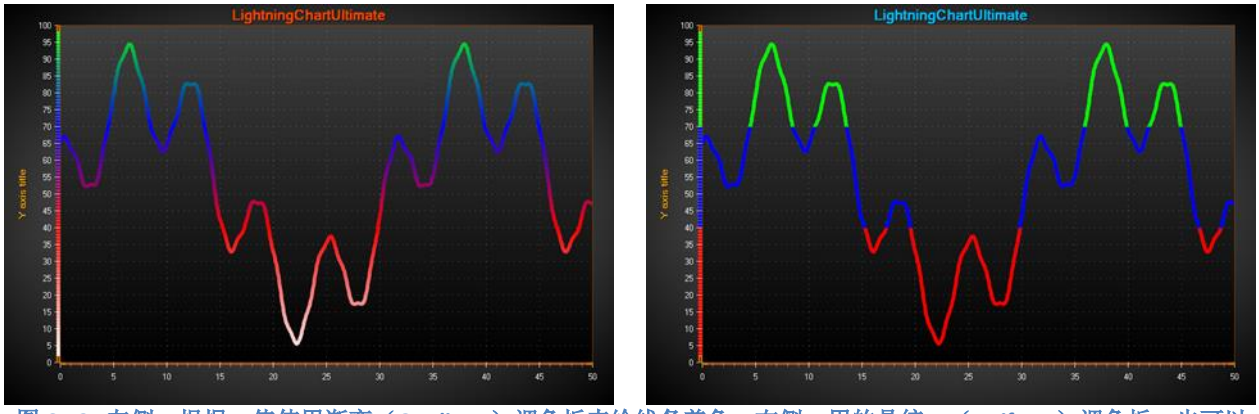


图 6-48. 左侧，根据 Y 值使用渐变（Gradient ）调色板来给线条着色。右侧，用的是统一（Uniform）调色板。也可以对 Y 轴启用 UsePalette

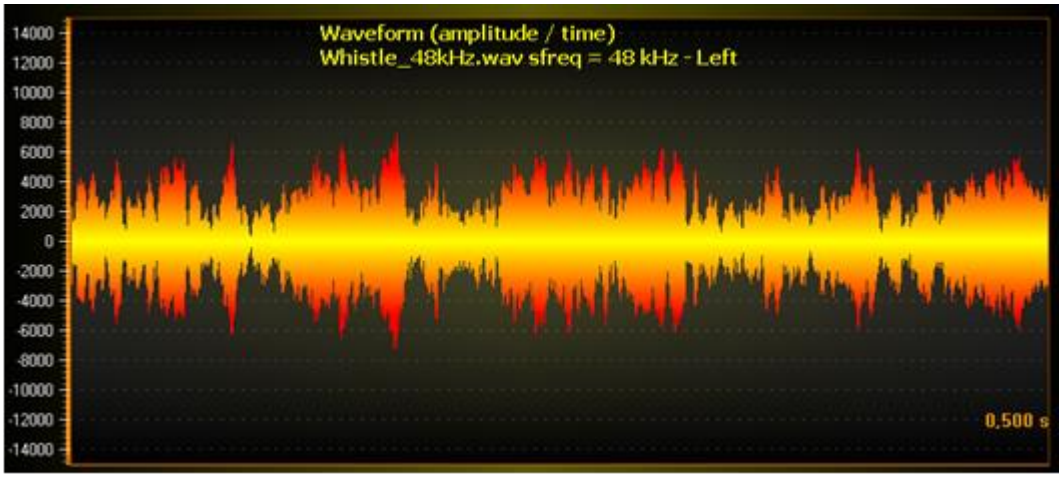


图 6-49. 用渐变（Gradient ）调色板为双极信号着色。对 Y 轴禁用 UsePalette。

6.14.2 通过 CustomLinePointColoringAndShaping 事件自定义外形并配色

用 ***CustomLinePointColoringAndShaping*** 事件，即在刚进入图表的渲染阶段就调用，可以自定义着色并调整坐标。

所有的线条模式（包括 Dash 和 Dot 等）都可以进行自定义着色。但是，只有当设置 ***LineStyle.Pattern = Solid*** 后，才可进行渐变着色。这在矢量文件导出方面也有严格的限制。

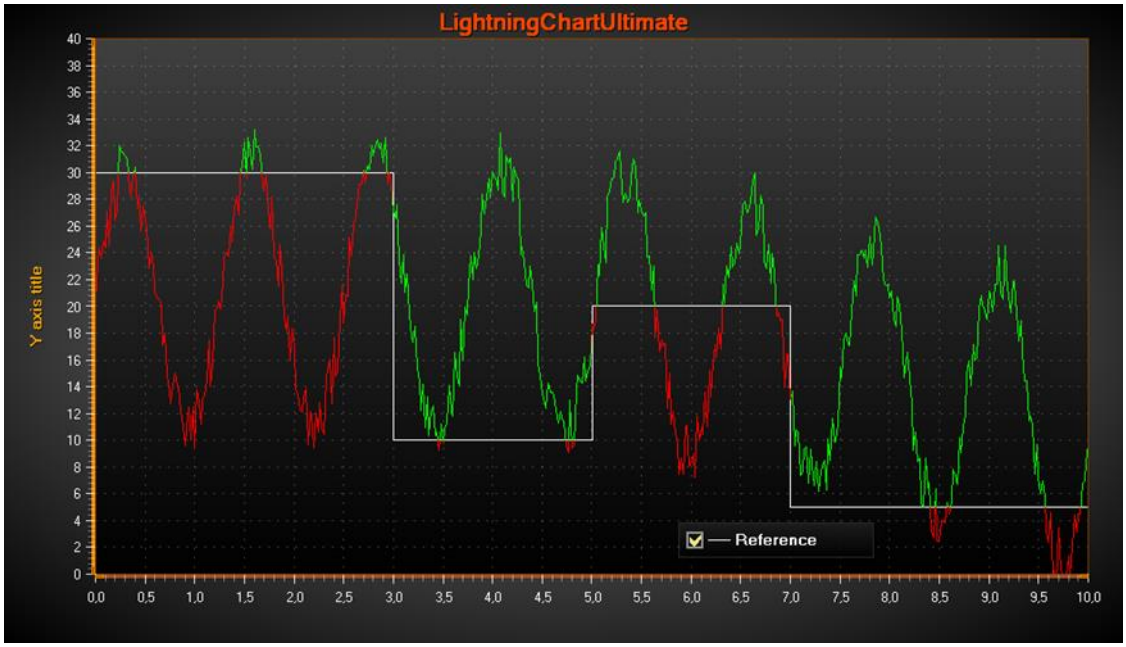


图 6-46. CustomLinePointColorAndShaping 事件处理程序，用于通过特定的变更参考水平更改线条颜色。

事件参数有以下信息：

- **CanModifyColors**：可以修改颜色。
- **Colors**:用LineStyle颜色预填充颜色数组。如果设置**CanModifyColors**为true，可以通过为预填充的颜色分配新值或创建一个新的颜色数组来进行修改。如果设置**CanModifyColors**为，则不填充。
- **CanModifyCoords**: 可以修改坐标。
- **Coords**: 预填充屏幕坐标数组。如果设置**CanModifyCoords**为 true，可以通过为预填充的颜色分配新值或创建一个新的颜色数组来进行修改。新的数组长度不必等于预先填充的长度。**退出事件处理程序时，确保Coords和Colors数组的长度相等。**如果**CanModifyCoords**设置为 false，则不填充。
- **HasDataPointIndices**：仅适用于 ***FreeformPointLineSeries***。
- **DataPointIndices**：包含在坐标和颜色数组中的数据点索引。如果线条结构中的后续点的X和Y值或坐标相等，则该图表将跳过它们。使用DataPointIndices信息，例如，可以从数据点的**PointColor**字段或外部颜色数组中为线点系列选择颜色。
- **SweepPageIndex**：如果***XAxis.ScrollMode*** = ***'Sweeping'***，则显示页面索引（0 或1）.

6.15 多项式回归

演示示例：*Regression Fit, Spline Line*

数据点的回归拟合仅适用于 PointLineSeries。 系列的回归拟合允许在线拟合和多项式拟合之间进行选择。 在后一种情况下，可以使用 RegressionPolyOrder 设置回归的程度。 当 RegressionFitting 设置为非 None 选项、线和点时替换为拟合曲线。

对于其他系列，可以使用 MathRoutines.PolynomialRegression() 方法并替换数据拟合点。

6.16 High-lowSeries （高低系列）

演示示例： High-Low; Stacked area; Stock course with previous close; Areas /high-lows; Scale breaks

高低系列将数据表示为高值和低值之间的填充区域。通过向 ***HighLowSeries*** 列表内添加 ***HighLowSeries*** 对象可以向图表中添加系列。

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

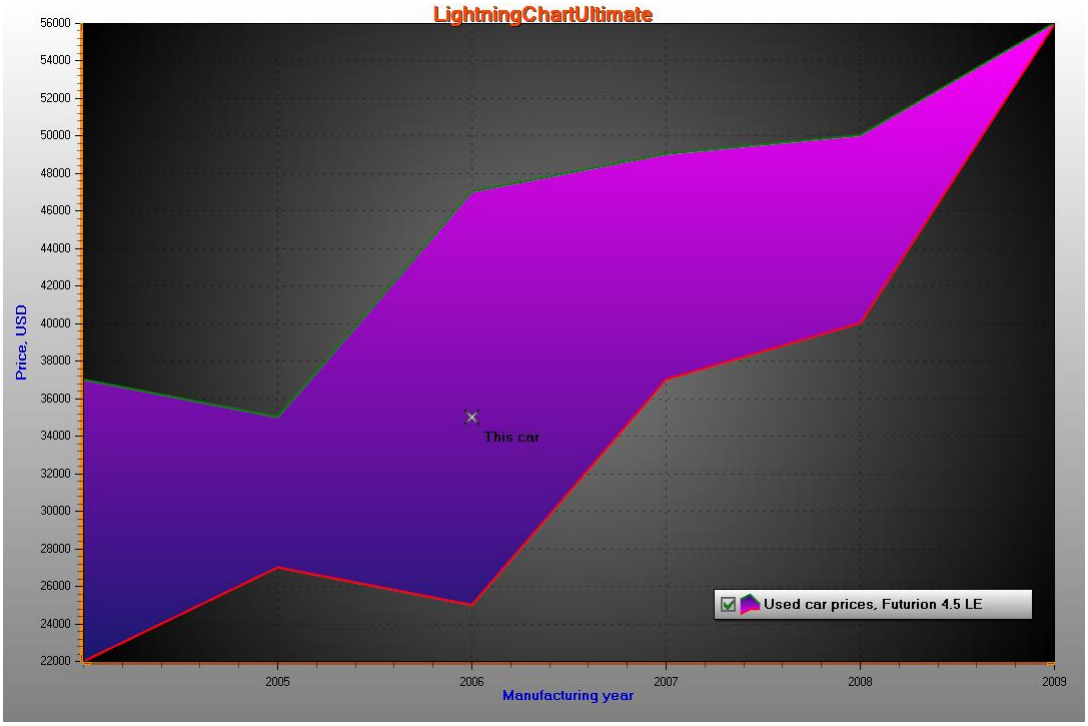


图 6-51. 上方 带有标记的高低系列。

6.16.1 填充、线和点样式

用 ***Fill*** 属性与其子属性可以设置填充效果。用 ***LineStyleHigh*** 和 ***LineStyleLow*** 属性可以定义线条样式。如果要让线条不可见，则分别设置 ***LineVisibleHigh = false***，以及 ***LineVisibleLow = false***。用 ***PointStyleHigh*** 和 ***PointStyleLow*** 属性可以定义点的样式。如果要让点不显示，则设置 ***PointsVisibleHigh = false***, ***PointsVisibleLow = false***。

具体线和点的样式细节，可以参阅第 6.6.1 章和 **Error! Reference source not found.** 章。当数据的高值小于其低值时，对该部分应用反向填充。用 ***ReverseFill*** 属性编辑逆反填充。

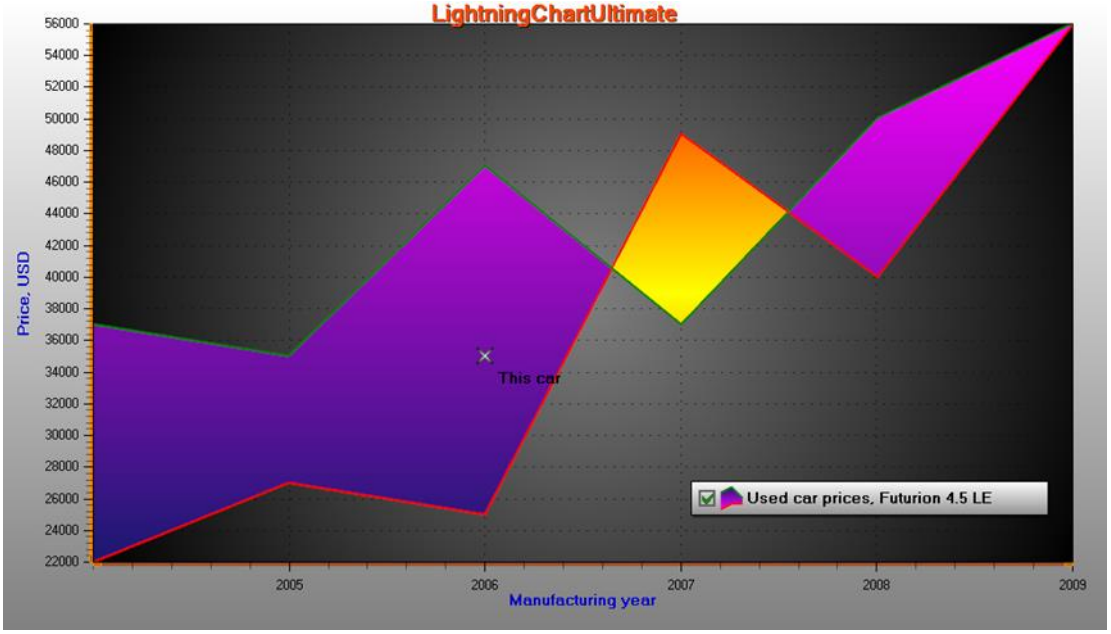


图 6-52. 第四个数据项逆反显示：高值 < 低值

6.16.2 Limits（界限）

启用 ***UseLimits*** 后，系列在 *exceed limit* 之上和 *deceed limit* 之下显示不同的纯色着色。接着，普通 ***Fill*** 和 ***ReverseFill*** 只作用于界限之间的范围。

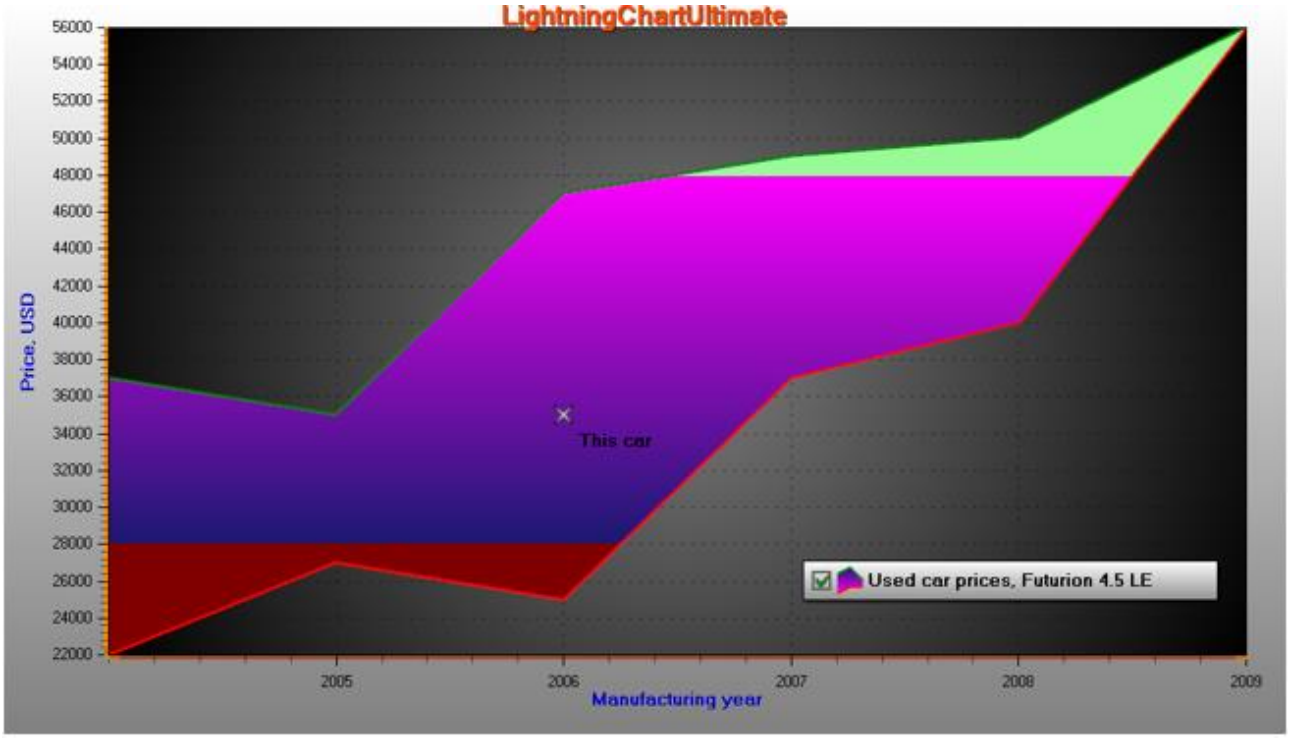


图 6-53. UseLimits = true, ExceedLimit = 48000 and DeceedLimit = 28000.

6.16.3 通过值域调色板着色

开启 ***UsePalette*** 后，填充采用 ***ValueRangePalette*** 分阶完成，也支持 ***Uniform*** 和 ***Gradient*** 着色。

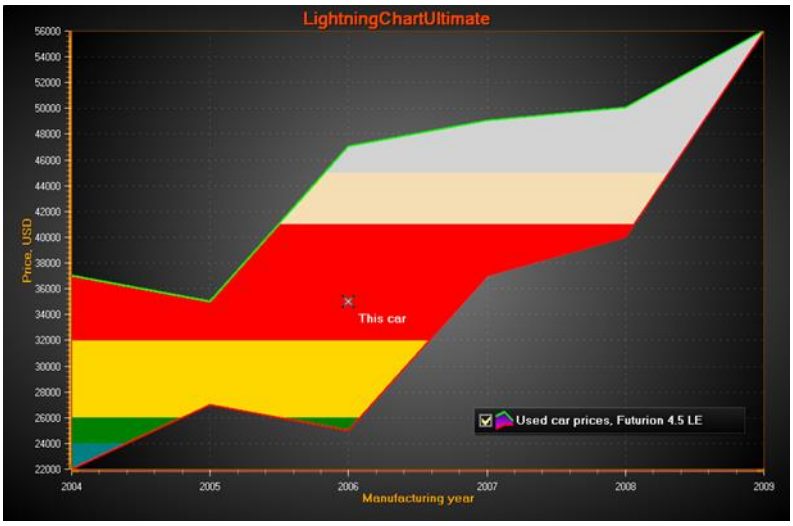


图 6-54. UsePalette = True，在 ValueRangePalette 中定义几个色阶。采用统一（Uniform）着色

6.16.4 添加数据

数据值必须在代码中进行添加。数据必须按 X 值的升序给出，**Points[i+1].X ≥ Points[i].X**。

使用 ***AddValues（HighLowSeriesPoint[], bool invalidate）***方法将数据值添加到现有值数组的末端。

```
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）;

//在末端添加数据
chart.ViewXY.HighLowSeries[0].AddValues（dataArray, true）;
```

要在覆盖旧数据的同时一次性设置全部系列数据，可以直接分配新数据数组：

```
//将数据分配到点数组中
chart.ViewXY.HighLowSeries[0].Points = dataArray;
```

6.17 AreaSeries

演示示例： Area; Areas; Data breaking in series; Multiple legends; Custom axis ticks

区域系列将数据表示为基面和值之间的填充区域。区域系列与在第 6.11 章介绍的 ***HighLowSeries*** 非常相似，但是更为简单。通过在 ***AreaSeries*** 列表中添加 ***AreaSeries*** 对象，可以向图表中添加系列。

```
chart.ViewXY.AreaSeries.Add（areaSeries）; //向图表添加区域系列
```

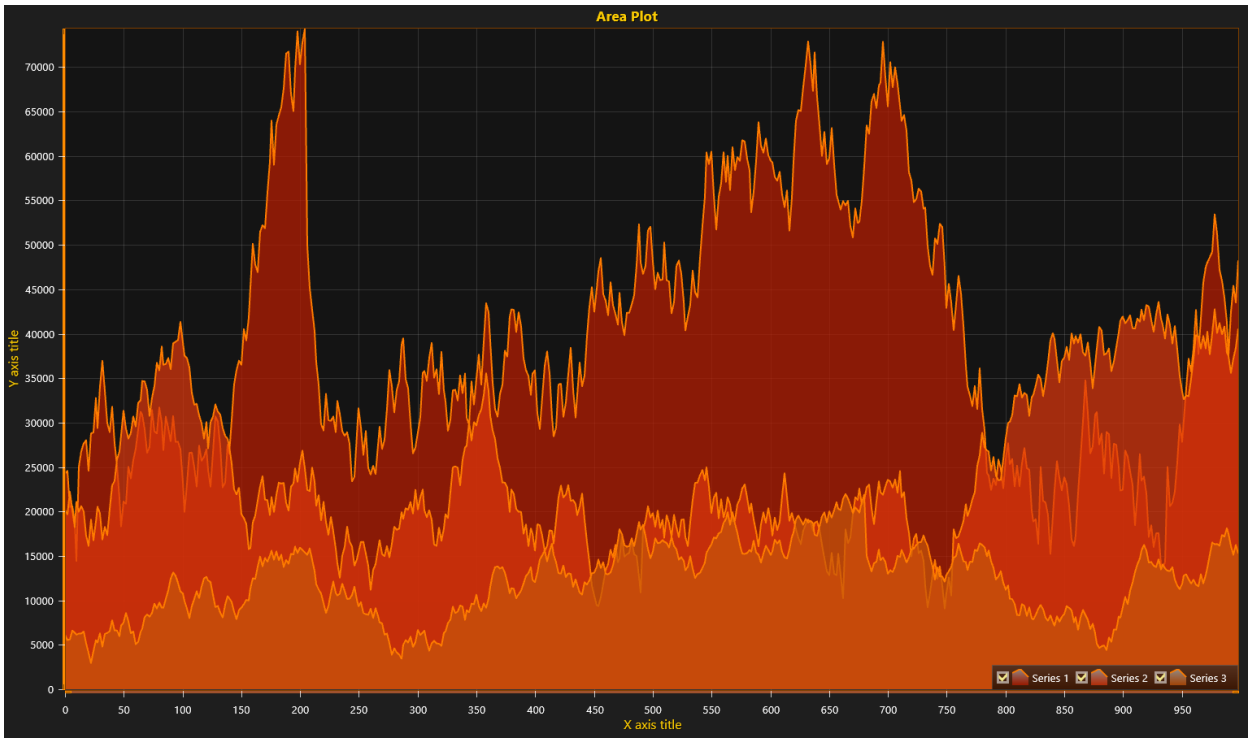


图 6-55. 三个区域系列均为 BaseValue = 0.

用 ***BaseValue*** 属性设置基面。用 ***Fill*** 属性设置首选填充样式。用 ***LineStyle*** 属性和 ***PointStyle*** 属性可以分别设置线条样式和点的样式。Exceed limits 和 deceed limits 可以像在 ***HighLowSeries*** 中一样应用。

6.17.1 添加数据

数据值必须在代码中添加。数据必须按 X 值的升序给出，**Points[i+1].X ≥ Points[i].X。**

使用 ***AddValues（AreaSeriesPoint[], bool invalidate）***方法将数据值添加到现有值数组的末端。

```
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）;

//在末端添加数据
chart.ViewXY.AreaSeries[0].AddValues（dataArray, true）;
```

要在覆盖旧数据的同时一次性设置全部系列数据，可以直接分配新数据数组：

```
//将数据分配到点数组中
chart.ViewXY.AreaSeries[0].Points = dataArray;
```

6.18 BarSeries （柱状系列）

演示示例： Vertical; Horizontal; Negative values; Stacked Bars

BarSeries 可以以水平或垂直的柱状来显示数据。

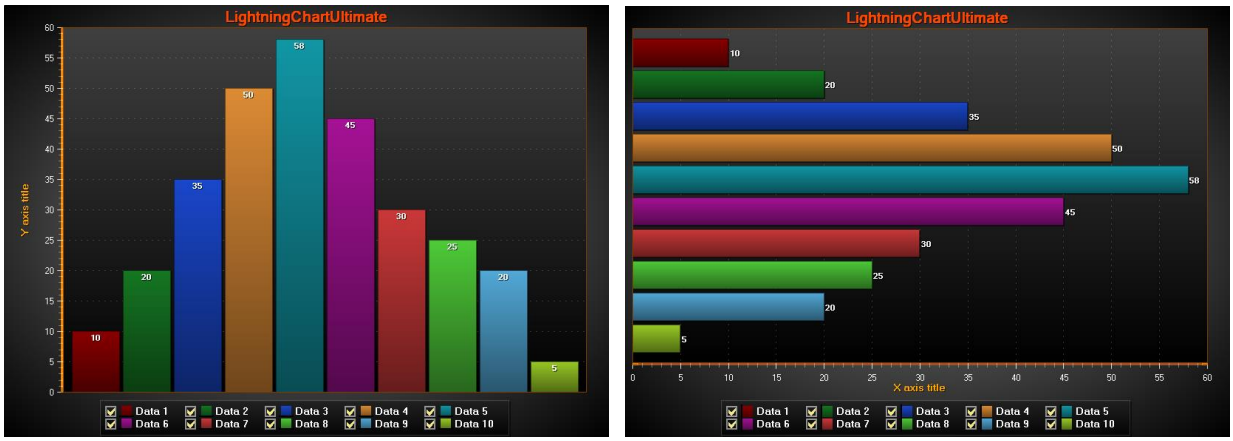


图 6-476.垂直与水平的柱状系列

用 ***Values*** 数组属性可以存储柱状数组的值。用 ***AddValue（…）*** 方法可以添加值。根据给定值索引用 ***SetValue（…）*** 方法来更新现有的值。值的类型为 ***BarSeriesValue***，具有以下字段：

- ***Value*** 柱状的长度
- ***Location*** 柱状在 X 轴的位置（垂直方向的外观）或 Y 轴位置（水平方向的外观）
- ***Text*** 在柱状内呈现的文本

用柱状系列的 ***LabelStyle*** 属性可以控制柱状值标签在图表上的显示方式。用 ***AddValue（…）*** 或 ***SetValue（…）*** 方法参数可以设置标签值文本。设置 ***Fill*** 属性及其子属性可以使用各种填充样式 。

用图表的 ***BarViewOptions*** 属性可以控制柱状显示的方式。用 ***BarView.Options.Orientation*** 可以在水平和垂直柱状方向二者之间进行选择。

BarViewOptions.Grouping 可以按值索引、按使用宽度拟合的索引或按位置值对柱状进行分组。它从外观视觉上把不同的柱状系列的值结合在一起。如果不需要分组的话，可以用

BarViewOptions.Grouping.ByLocation，并卫每个 BarSeriesValue 对象设置不同的 ***Location*** 字段。使用宽度拟合属性来调整列之间和旁边的空间。当不使用宽度拟合时，柱状系列的 ***BarThickness*** 属性决定了柱状的宽度。设置 ***BarViewOptions.Stacking*** 为 ***Stack*** 或 ***StackStretchToSum*** 可以将组进行堆叠。当使用 ***StackStretchToSum*** 时，通过设置 ***StackSum*** 属性可以定义目标总数。默认情况下用 100 表示 100%。

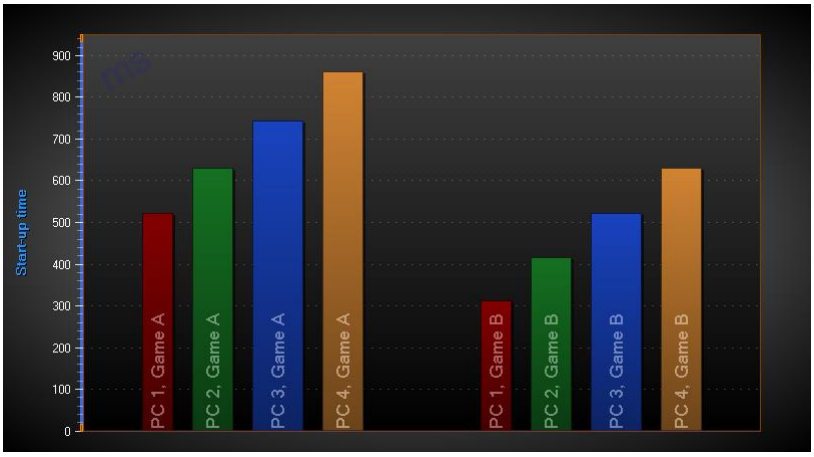


图 6-487 柱状系列 Grouping = ByIndex, Stacking = None.

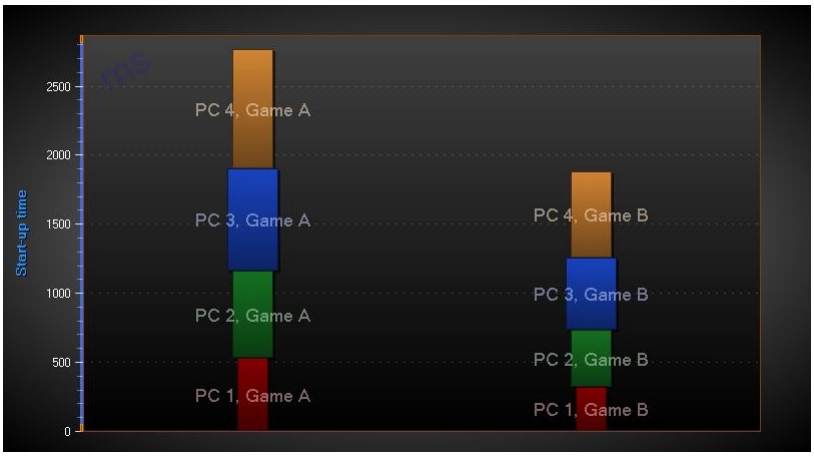


图 6-58. 柱状系列 Grouping = ByIndex, Stacking = Stack.

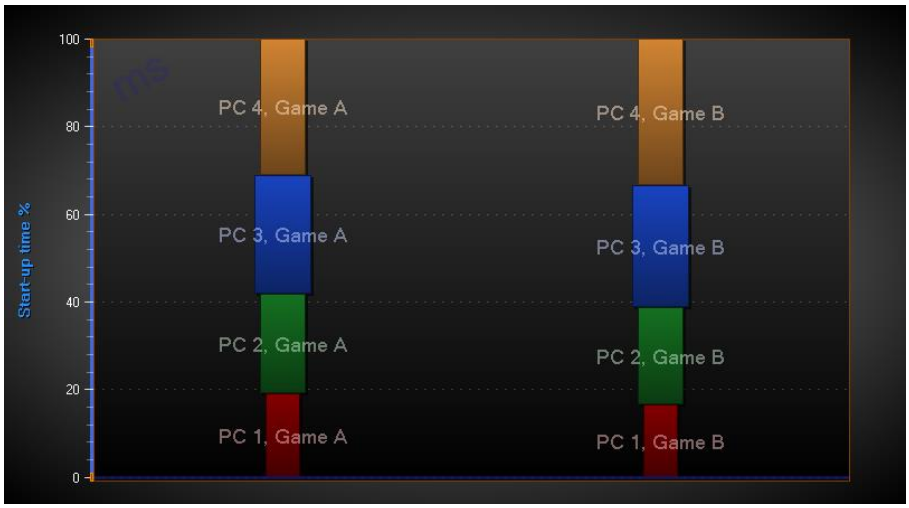


图 6-59. 柱状系列 Grouping = ByIndex, Stacking = StackStretchToSum. StackSum = 100.

在 ***BarSeries*** 中的 ***BaseLevel*** 属性是所有值中的最小值，并设定了柱状的起始位置。在 ***Stacked*** 视图中，它可以增加（若为正）或减少（若为负）柱状的尺寸。在 ***StackedToSum*** 视图中，柱状的尺寸是成比例的，并如同 ***Stacked*** 中一样进行计算。

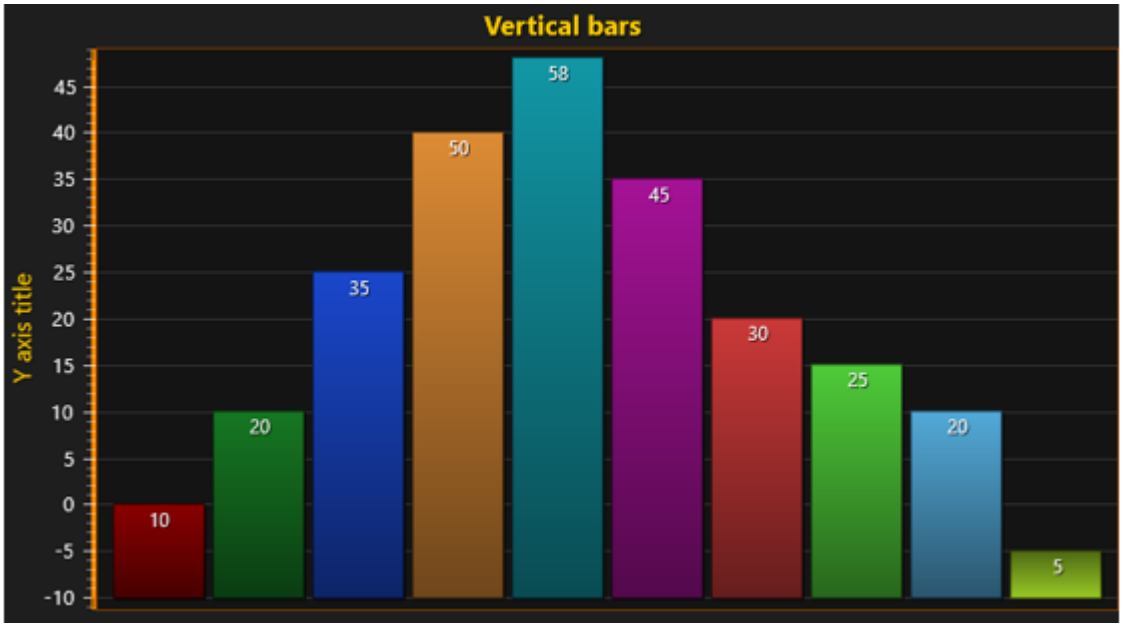


图 6-60. BaseLevel 设置为 -10；条性值分别为： 10、20、35、 50、 58、45、30、25、 20、 5。

6.19 StockSeries 股票系列

演示示例： Segments with splitters; Stocks and bars; Scale breaks; Statistic analytics

股票系列可以把股票交易数据以蜡烛图或股票柱状图格式可视化。通过在 ***StockSeries*** 列表属性中添加几个 ***StockSeries*** 对象，可以在同一个图表中添加多个股票系列。用 Style 属性可以选择样式。样式选项有： ***Bars***、 ***CandleStick*** 和 ***OptimizedCandleStick***。

用 ***ColorStickDown***、***ColorStickUp***、***FillDown*** 和 ***FillUp*** 属性来设置着色和填充选项。用 ***StickWidth*** 属性以像素为单位调整柱状的宽度。用 ***ItemWidth*** 属性调整总数据项宽度。

为了获得最大的渲染性能，可以使用 ***Bars*** 样式，设置 ***StickWidth*** = 1。

使用 ColorStickDown、ColorStickUp、FillDown 和 FillUp 属性设置着色和填充选项。

使用 StickWidth 属性调整棒的宽度（以像素为单位），使用 ItemWidth 属性调整数据项的总宽度。 通过设置 Behind = True，可以将 StockSeries 设置为在线条系列之前渲染。***StockSeries*** 可以设置为在线条系列之前渲染，即设置 ***Behind = True***。

```
// Modifying StockSeries properties.
stockSeries.Style = StockStyle.OptimizedCandleStick;
stockSeries.ItemWidth = 13;
stockSeries.StickWidth = 3;
stockSeries.Behind = false;
```

StockSeries 还具有数据打包属性，当启用该属性时，会导致将彼此接近的数据值打包到单个渲染项中。 这提高了性能，尤其是对于较大的数据集，但数据可能不如没有打包的情况下准确.

```
// Enabling data packing.
stockSeries.Packing = StockSeriesPacking.On;
```

图 6-62. 设置 StockSeries 样式 Style = CandleStick. 淡蓝色的线是后面的一个 PointLineSeries，贯穿所有收市值。

图 6-63. 设置 StockSeries 样式 Style = Bars. 线系列用来显示线性回归拟合和该线的偏移量（2 *标准偏差）。用一条带形为线性拟合选择一个日期范围。

6.19.1 StockSeries 的数据设置

创建一个数据数组并设置数组项。每个项具有以下字段：

Date	DateTime值（年、月、日）
Open	当天的开市值
Close	当天的收市值
Low	当天的最低值
High	当天的最高值
Transaction	总交易金额 (可选的)
Volume	股票交易数量 (可选的)

始终按 Date 值（最早的日期排第一）以升序来保存数据。

```
// 创建数据数组
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）
};
  //将数据数组分配给各个系列
  chart.ViewXY.StockSeries[0].DataPoints = data;
```

6.19.2 设置 X 轴显示日期

```
chart.ViewXY.XAxes[0].ValueType = AxisValueType.DateTime;
chart.ViewXY.XAxes[0].LabelsAngle = 90;
chart.ViewXY.XAxes[0].LabelsTimeFormat =
  System.Globalization.CultureInfo.CurrentCulture.DateTimeFormat
    .ShortDatePattern;
chart.ViewXY.XAxes[0].MajorDiv = 24 * 60 * 60; //主要分度是以秒为单位的一天
chart.ViewXY.XAxes[0].AutoFormatLabels = false;

//设置日期起点
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;
```

设置适合数据的 X 轴范围：

```
// 在x轴两端延伸半天。使用第一个和最后一个日期值
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）;
```

6.19.3 自定义外观格式化

StockSeries 有 ***CustomStockDataAppearance*** 事件处理程序，可用于单独格式化系列数据项的外观，用属性覆盖通用的填充和颜色样式。在事件处理程序中，修改特定点的宽度和颜色。

图 6-63. 设置 CustomStockDataAppearance 可以用更大宽度和更亮的渐变颜色来突出显示特定数据项。

6.19.4 应用 Scale breaks（刻度中断）

要剪掉无交易活动的小时与天，可参阅第 6.3.2 章节。

6.20 PolygonSeries 多边图

演示示例： Polygons; Box-whisker plot; Ternary plot; Image viewer; Zoomable 2D pie

用 ***PolygonSeries*** 可根据给定的边界路径来渲染填充和边界线。

在 ***Fill*** 属性中设置填充性能。用 ***PolygonSeries*** 的 ***Border*** 属性设置边界线条样式。

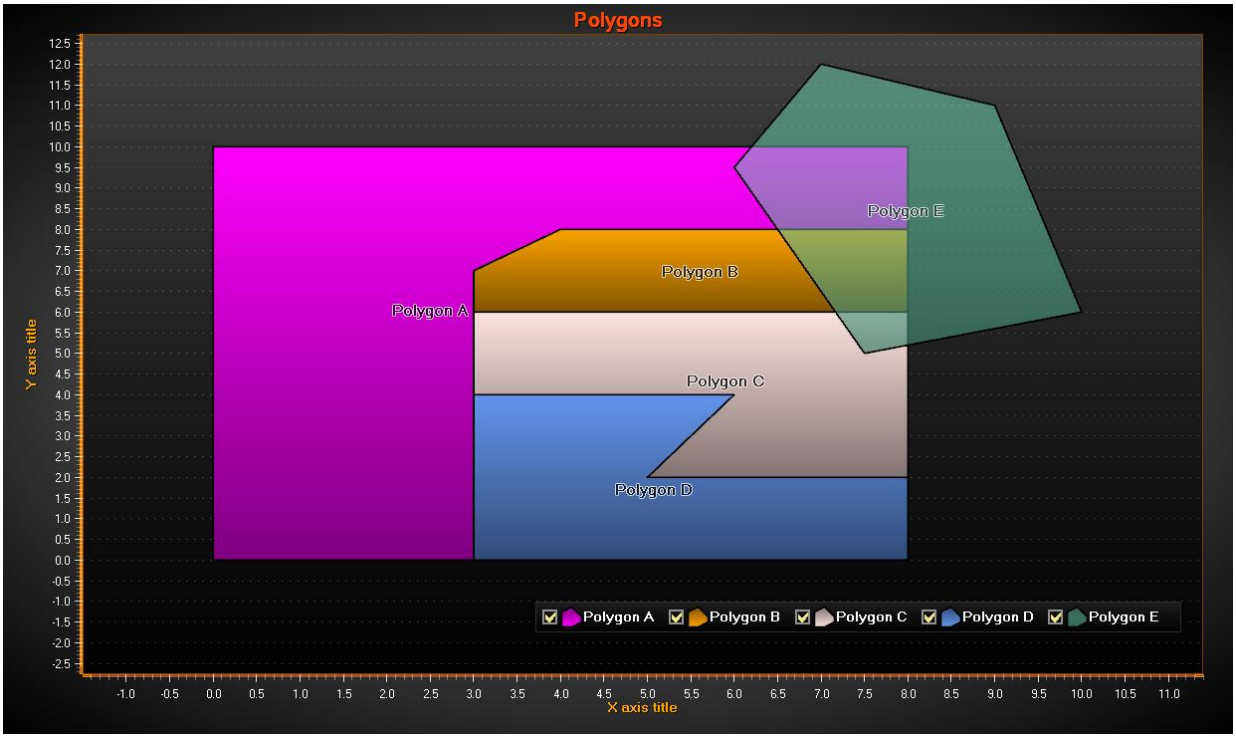


图 6-6449.几个多边形。

6.20.1 为多边形设置数据

在 ***Points*** 属性中设置路径点。 ***PolygonSeries*** 具有一种一个自动路径关闭的功能，即如果最后一个点没有连接到第一个点，图表将会自动完成连接。

下面展示了如何分配上面一张图片的透明青色多边形的路径点：

```
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）};
```

6.20.2 启用复合/相交填充

设置 ***IntersectionsAllowed = True*** 使多边形路径自身相交。若没有启用该属性，当路径相交时，填充会出现完全错乱。因为检测和渲染相交的情况非常频繁，出于性能原因，默认情况下，该属性设置为 ***False*** 。

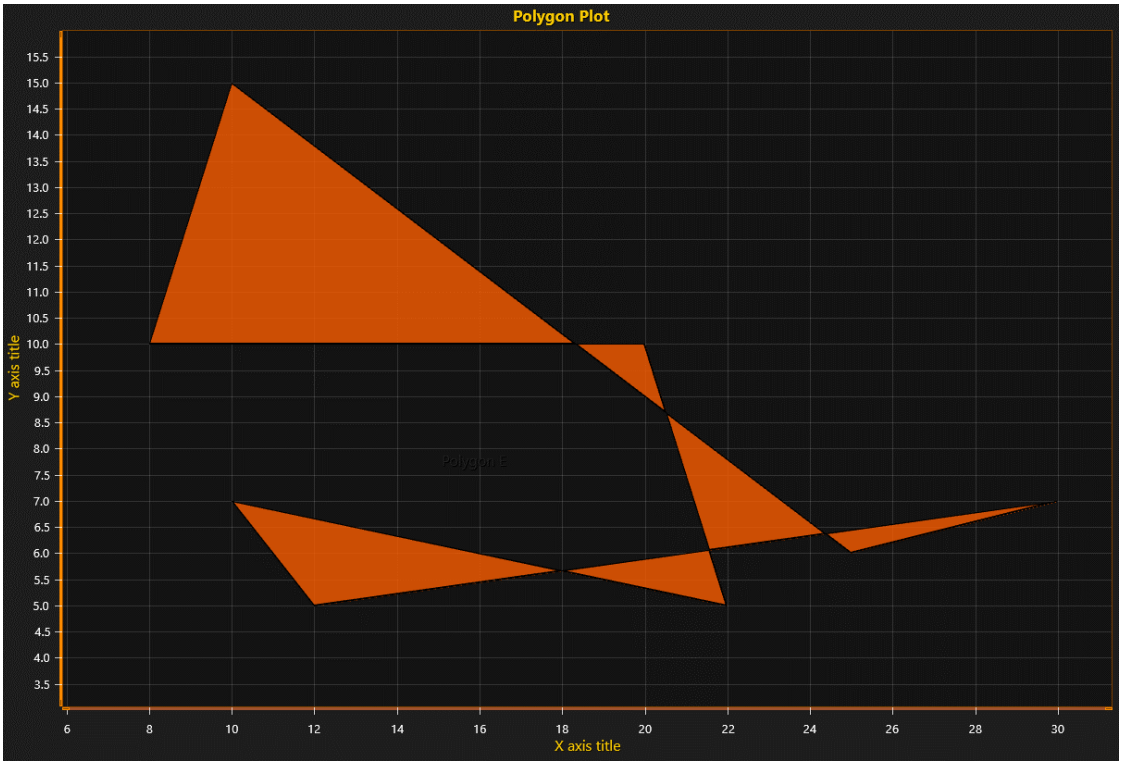


图 6-65. 路径相交的多边形，设置 IntersectionsAllowed = True.

6.21 LineCollections （线集）

演示示例： Line Collections; Line spectrogram; Stem plot

LineCollection 是一些线段的集合。每个线段都是一条从 A 点到 B 点的线。一个 ***LineCollection*** 可包含数千条线段。与 ***PointLineSeries***、***FreeformPointLineSeries*** 或 ***SampleDataSeries*** 相比而言，***LineCollection*** 在渲染数千条不同的线段方面非常有效。***PointLineSeries***、***FreeformPointLineSeries*** 或 ***SampleDataSeries*** 在渲染由成千上万个点构成的连续多线段时的效率更高。

用 ***LineStyle*** 属性控制线条颜色、样式和宽度。在 ***Lines*** 属性中设置线段。

在 ***ViewXY.LineCollections*** 列表属性中添加 ***LineCollection*** 对象。

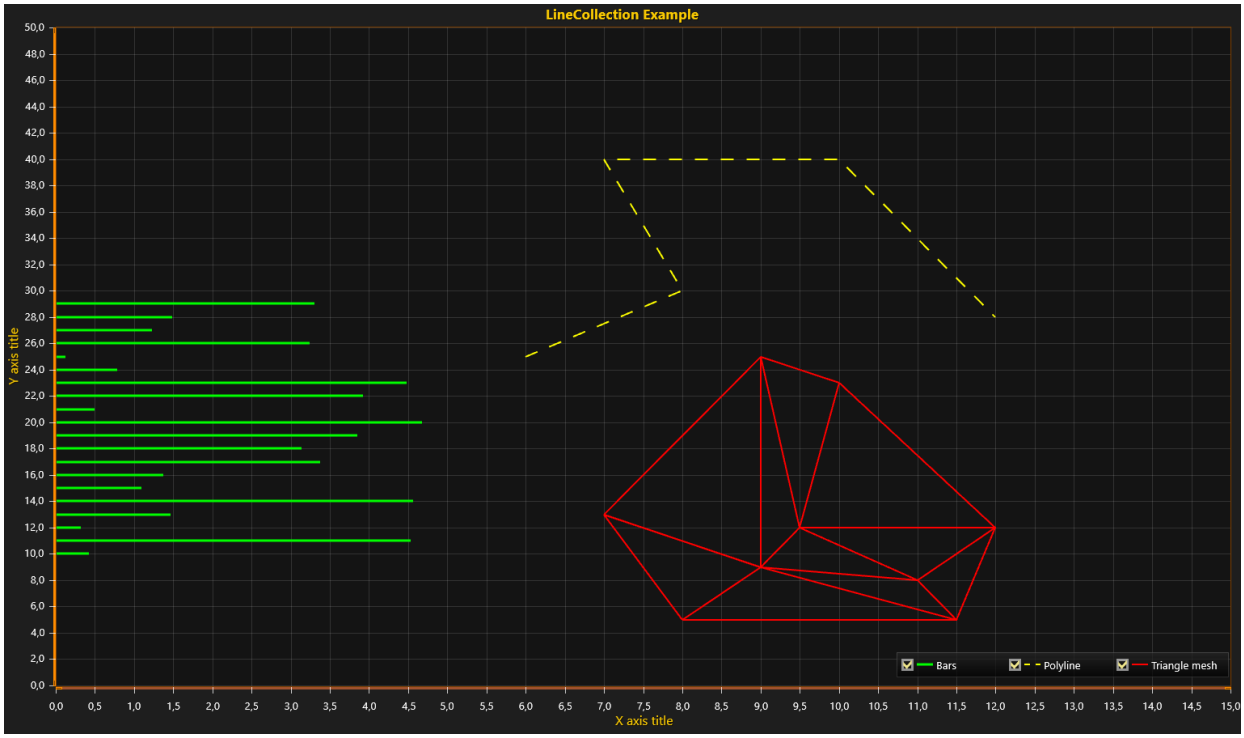


图 6-66. 三个应用中的 LineCollections；绿色是非常快速渲染的柱状，黄色为多段线，红色是任意三角形线框网格

6.21.1 为 LineCollection 设置数据

SegmentLine 结构包含四个字段：

AX 起始点， X
AY 起始点， Y
BX 终点， X
BY 终点，Y

将 ***SegmentLines*** 数组添加到 ***Lines*** 属性，方法如下：

```
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）};
```

6.21.2 解决单个分段

GetSegmentsAtPoint（）方法允许检查哪个单个分段在给定位置，例如

在鼠标坐标下的示例。 它返回一个整数列表（segment line indexes）。

List <int> list = _chart.ViewXY.LineCollections [0] .GetSegmentsAtPoint（xCoordinate，

yCoordinate）；

6.22 IntensityGridSeries （强度网格）

演示示例： Heat map; Spectrogram; Intensity grid mouse control

IntensityGridSeries 能够可视化 M x N 的节点数组，并通过指定的值域调色板着色。节点之间的颜色采用的是插值法。***IntensityGridSeries*** 是在 X 和 Y 轴方向上均匀分布的矩形系列，可以渲染等高线、等高线标签以及线框图等。

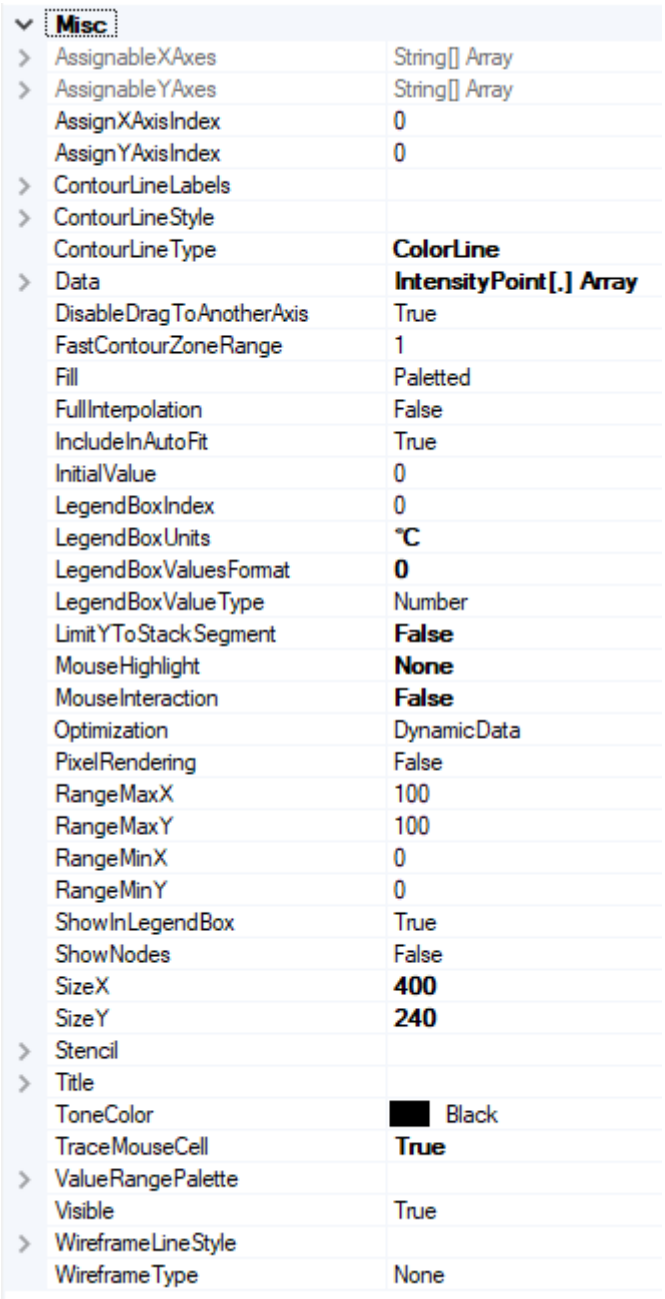


图 6-67. IntensityGridSeries 属性.

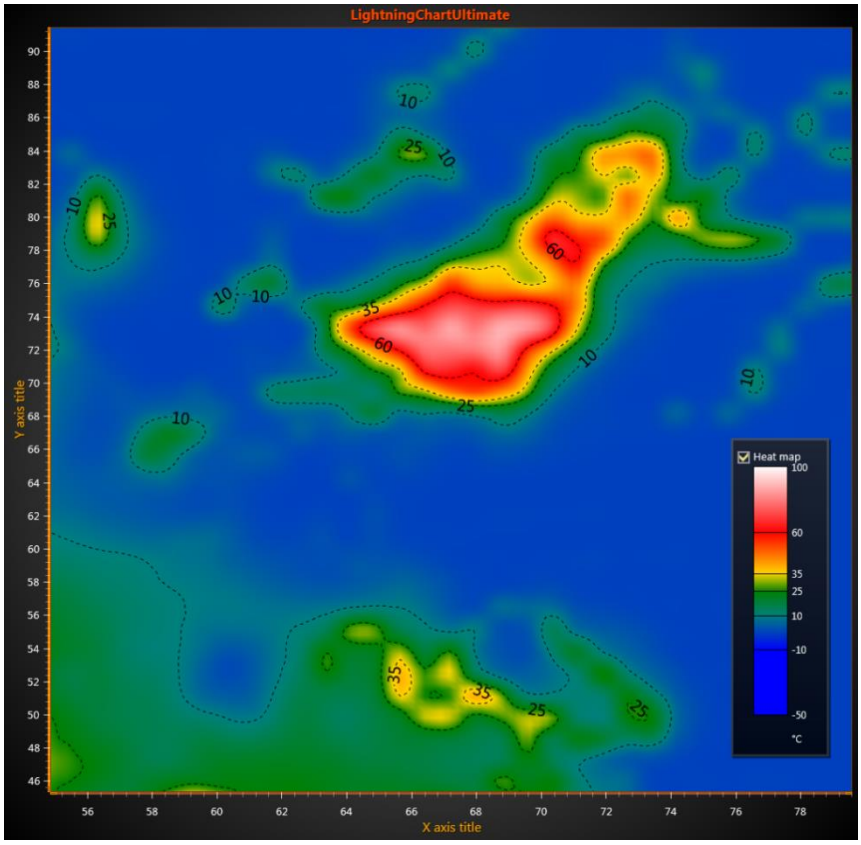


图 6-68. IntensityGrid 系列展示了一幅热度图示。图例框显示了值域调色板。

数据以二维数组的形式存储在 ***Data*** 属性中。每个数组项的类型都为 ***IntensityPoint***。每个节点的数据值存储在 ***IntensityPoint*** 结构的 ***Value*** 字段中，从而表明应使用 ***ValueRangePalette*** 中的那种颜色。

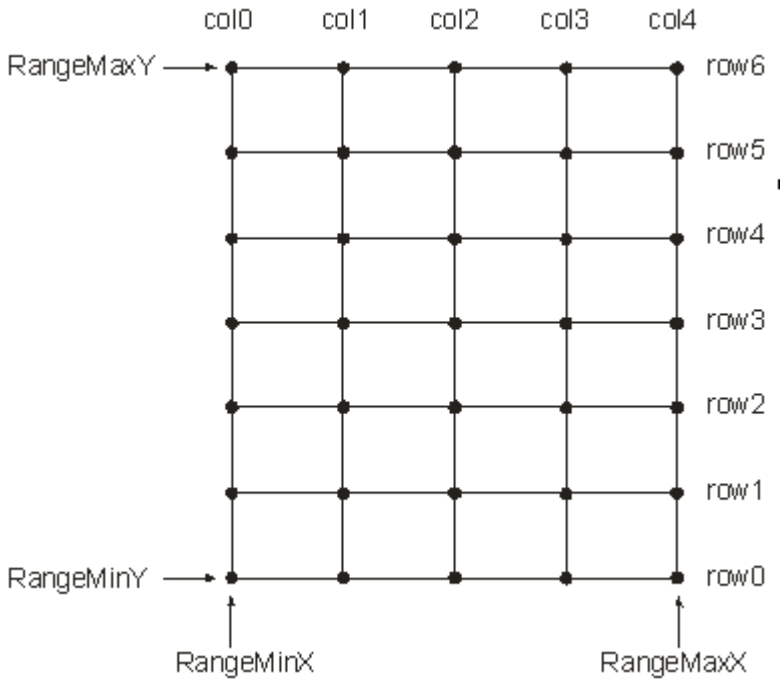


图 6-69. IntensityGridSeries 节点； SizeX = 5, SizeY = 7.

节点距离自动计算，公式如下

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

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

6.22.1 设置强度网格数据

- 用 ***RangeMinX*** 和 ***RangeMaxX*** 属性设置 X 范围，对指定的 X 轴的最大值和最小值进行排序。
- 用 ***RangeMinY*** 和 ***RangeMaxY*** 属性设置 Y 范围，对指定的 Y 轴的最大值和最小值进行排序。
- 设置 ***SizeX*** 和 ***SizeY*** 属性，将网格的尺寸规格指定为列和行。
- 为每个节点设置 ***Value：***

用数据组索引的方法

```
for （int nodeIndexX = 0; nodeIndexX < columnCount; nodeIndexX ++）
{
        for （int nodeIndexY = 0; nodeIndexY < rowCount; nodeIndexY ++）
        {
          intensityValue =//一些高度值
          gridSeries.Data[iNodeX, iNodeY].Value = intensityValue;
        }
}
gridSeries.InvalidateData（）; //准备好新值后通知刷新
```

另一种使用 SetDataValue 的方法

```
for （int nodeIndexX = 0; nodeIndexX < columnCount; nodeIndexX ++）
{
        for （int nodeIndexY = 0; nodeIndexY < rowCount; nodeIndexY ++）
        {
          intensityValue =//一些高度值
          gridSeries.SetDataValue（nodeIndexX, nodeIndexY,
                    0, // X值与网格无关
                    0, //Y值与网格无关
                    intensityValue,
                    Color.Green）; //本例中没有使用源点颜色，所以这里用什么颜色都行
        }
}
gridSeries.InvalidateData（）; //准备好新值后通知刷新
```

仅对现有网格设置值

当数据快速改变，而 IntensityMesh 的几何形状，或者 IntensityGrid 系列的 SizeX 或 SizeY 没有变化时，使用 ***SetValuesData*** 方法是最好的。因为它可接受 Double[][] 格式的数据组，能够快速滚动或对行或列重新排序。特别是结合 ***PixelRendering*** 属性（参阅第 **Error! Reference source not found.**

章节）一起使用时，会是一种非常有效的高分辨率滚动光谱图可视化方法。注意，当使用 SetValuesData 设置的外部数据数组使 ***PixelRendering*** 处于 ***disabled*** 状态时，***Data*** 属性不能为空。

仅对现有网格设置颜色

当数据快速改变，而 IntensityMesh 的几何形状，或者 IntensityGrid 系列的 SizeX 或 SizeY 没有变化时，使用 ***SetColorsData*** 方法是最好的。因为它可接受 int[][]格式的值，例如 GPU 可直接接受的 ARGB 值。采用这种数据数列，能够快速滚动或对行或列重新排序。特别是结合 ***PixelRendering*** 属性（参阅第 **Error! Reference source not found.**章节）一起使用时，会是一种非常有效的高分辨率滚动光谱图可视化方法。注意，当使用 SetColorsData 设置的外部数据数组使 ***PixelRendering*** 处于 ***disabled*** 状态时，***Data*** 属性不能为空。

6.22.2 根据位图文件创建强度网格数据

演示示例：热度图（Heat map）

根据一幅位图图像创建一个面。可以用 ***SetHeightDataFromBitmap*** 方法来实现。通过系列 ***Data*** 数组属性可获得位图尺寸（如果没有使用反锯齿或重新采样）。就每个位图图像像素，分别对 Red、Green 和 Blue 值求和；和越大，该节点的数据值就越大；黑色和深色的值较低，而明亮和白色的值较高。

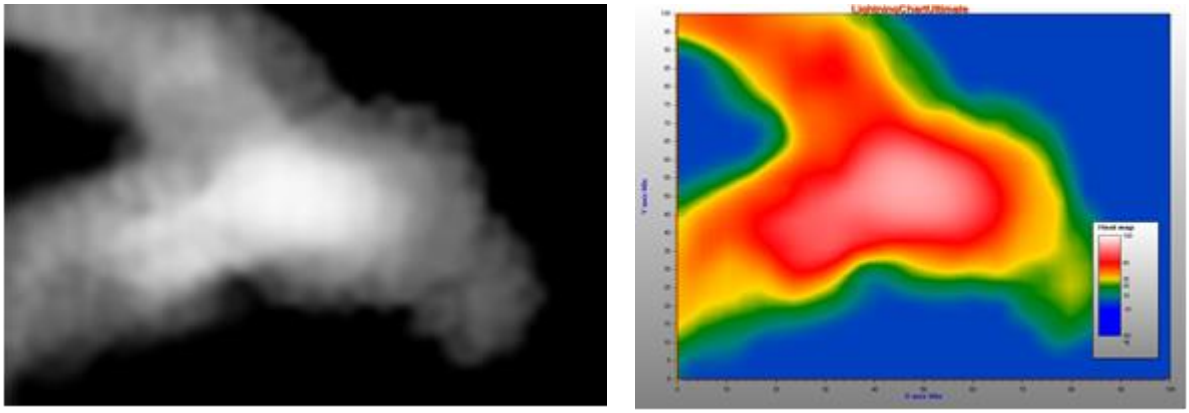


图 6-70. 源位图和计算所得的强度值数据。黑色区的值较低，明亮区的值较高。

6.22.3 填充样式

使用 ***Fill*** 属性来选择填充样式。有以下可选项：

- ***None***: 此选项，不填充任何样式。采用线框网格或普通轮廓线时使用此项。
- ***FromSurfacePoints***: 使用数据属性节点的颜色。
- ***Toned***: 应用 ToneColor
- ***Paletted***: 参阅第 6.16.5 章节。

开启 ***FullInterpolation*** 属性后，会在填充中使用增强型插值方法。注意这将更多地用到 CPU 和 GPU。使用全插值，填充效果会更好，但只能在数据数组大小相当小的情况下才能看到。

6.22.4 渲染为像素图

开启 ***PixelRendering*** 属性后，节点渲染为像素或矩形。这是一个非常高性能的渲染类型，例如实时高分辨率热成像应用。注意，当选择此渲染模式时，许多其他选项将禁用，如等高线、线框和插值。如果使用对数轴，对数变换只适用于系列的尖角，位图中的像素保持均匀的间隔，并且不进行对数变换。

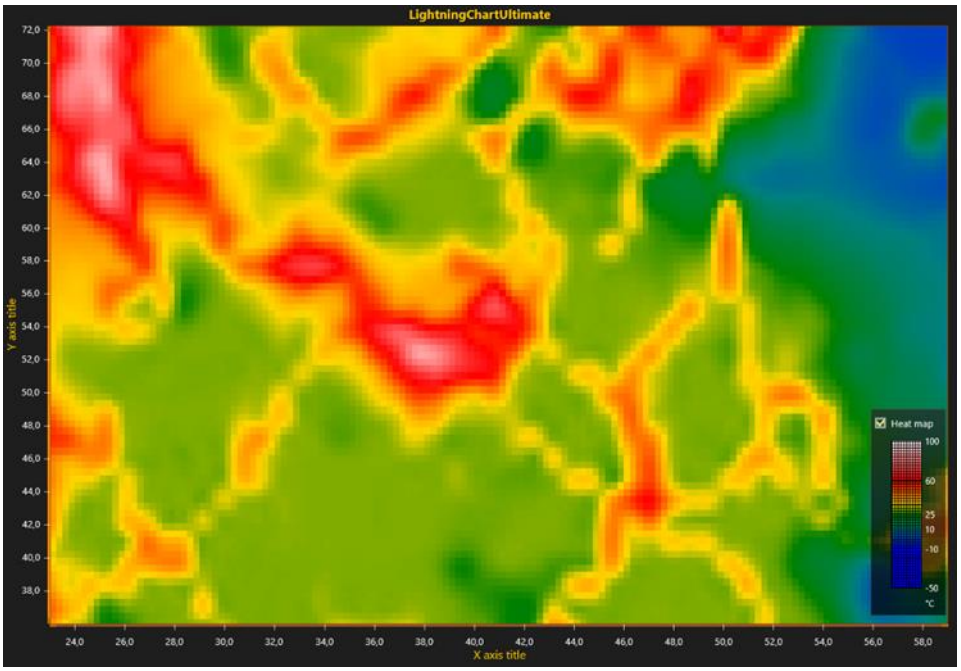


图 6-71. PixelRendering = true.

6.22.5 ValueRangePalette （值域调色板）

用 ***ValueRangePalette*** 属性，可为值的着色定义几个颜色阶。 ***ValueRangePalette*** 可用于：

- ***Fill*** （参阅第 **Error! Reference source not found.**章节）
- ***Wireframe*** （参阅第 6.22.6 章节）
- ***Contour lines*** （参阅第 6.22.7 章节）

为高度调色板定义几个阶。每一阶都有一个高度值和相对应的颜色。

注意! 可预编译并快速加载 20 个色阶。阶数越高，初始化图表时可能会有几秒钟的延迟。

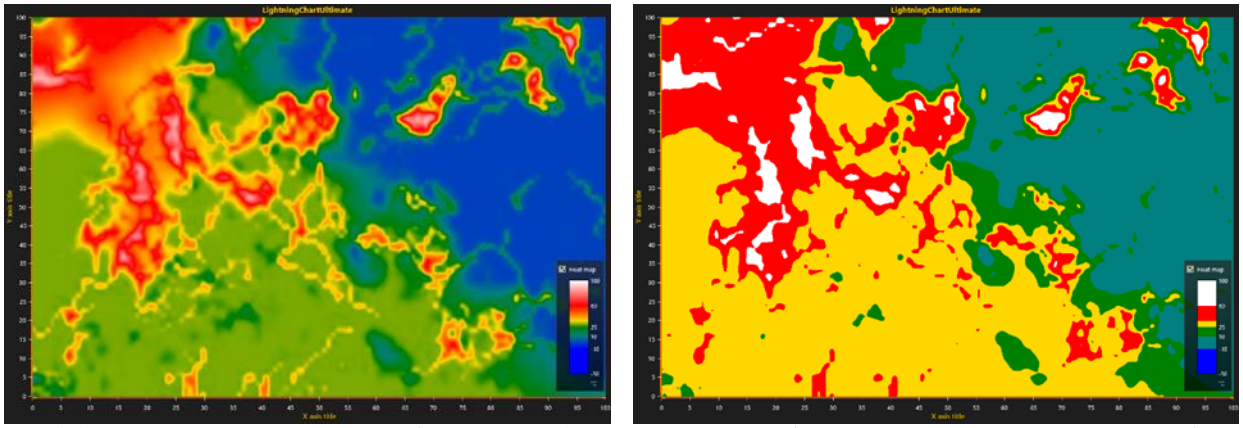


图 6-72 左侧， IntensityGridSeries Fill 设置为 Paletted，并且 Palette Type 设置为 Gradient；右侧， Palette Type 设置为 Uniform

调色板用 ***MinValue***、***Type*** 和 ***Steps*** 属性来定义。***Type*** 属性有两个选择： ***Uniform*** 和 ***Gradient***。前面的等高线调色板（ 查看图例框）显示为：

- ***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

第一个阶值以下的值使用第一个阶的颜色着色。

6.22.6 Wireframe （线框）

用 ***WireframeType*** 选择线框样式。选项有：

- ***None***: 无线框
- ***Wireframe***: 纯色线框。用 ***WireframeLineStyle.Color*** 设置颜色。
- ***WireframePaletted***：在 ***ValueRangePalette*** 中为线框着色（参阅第 6.16.5 章节）
- ***WireframeSourcePointColored***: 根据网格节点的颜色为线框着色
- ***Dots***: 在网格节点位置绘制实心色点
- ***DotsPaletted***: 在网格节点位置绘制点， 并通过 ***ValueRangePalette*** 着色
- ***DotsSourcePointColored***: 在网格节点位置绘制点， 根据网格节点的颜色来着色

线框的线条样式（颜色、宽度、图案）可以用 ***WireframeLineStyle*** 来编辑。

注意！ 仅当设置 ***WireframeLineStyle.Width = 1*** 且 ***WireframeLineStyle.Pattern = Solid*** 时， 调色板着色的线框线条和点才可用。

6.22.7 Contour lines 等高线

演示示例： Heatmap color spread; Contours with labels

设置填充和边框属性来使用等高线。通过设置 ***ContourLineType*** 属性，可以用不同的样式绘制等高线：

- ***None***: 不显示等高线
- ***FastColorZones***:线条绘制为细细的带状 。 适用于非常强的渲染，非常适合于持续更新或外形多样的曲面。陡峭值的变化以细线表示，而平缓坡度的高差以粗带表示。用 ***ContourLineStyle.Color*** 属性为每个线条定义采用同样的颜色。通过 ***FastContourZoneRange*** 属性可以设置带的宽度。值在 Y 轴范围内。
- ***FastPalettedZones***: 与 ***FastColorZones*** 类似，但是要在 ***ValueRangePalette*** 选项中为线条着色 参阅第 6.22.5 章节）。
- ***ColorLine***: 与 ***FastColorZones*** 类似，但是等高线是实际的线；渲染需要更长的时间，不建议持续更新或动画表面采用。 用 ***ContourLineStyle.Width*** 属性可以调整线条的宽度。
- ***PalettedLine***: 与 ***ColorLine*** 类似，但是要在 ***ValueRangePalette*** 选项中为线条着色。

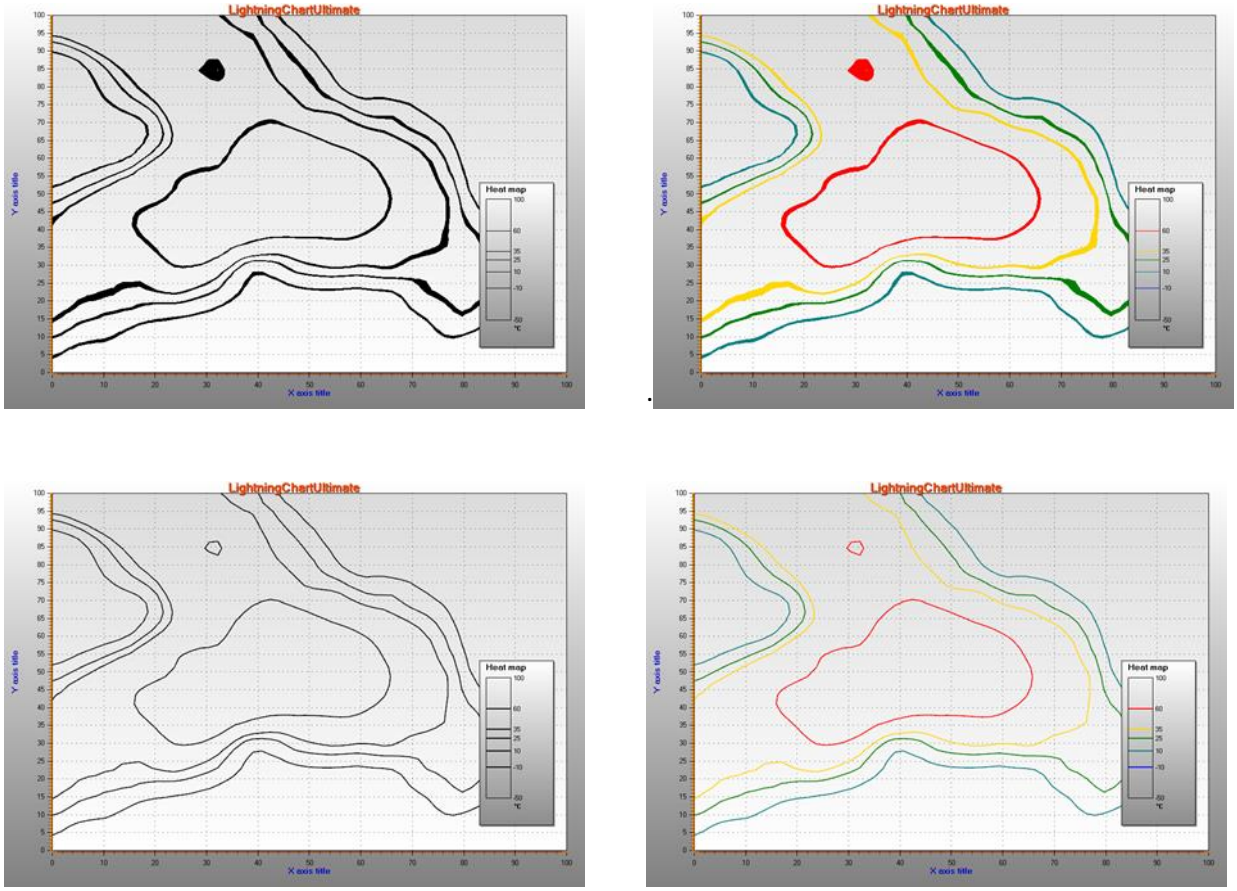


图 6-73. 左上角，ContourLineType = FastColorZones. 右上角，ContourLineType = FastPalettedZones

左下角，ContourLineType = ColorLine。 右下角，ContourLineType = PalettedLineContourLineType = ColorLine；

6.22.8 等高线标签

当等高线设置为可见时，可以在线路径上显示数值。

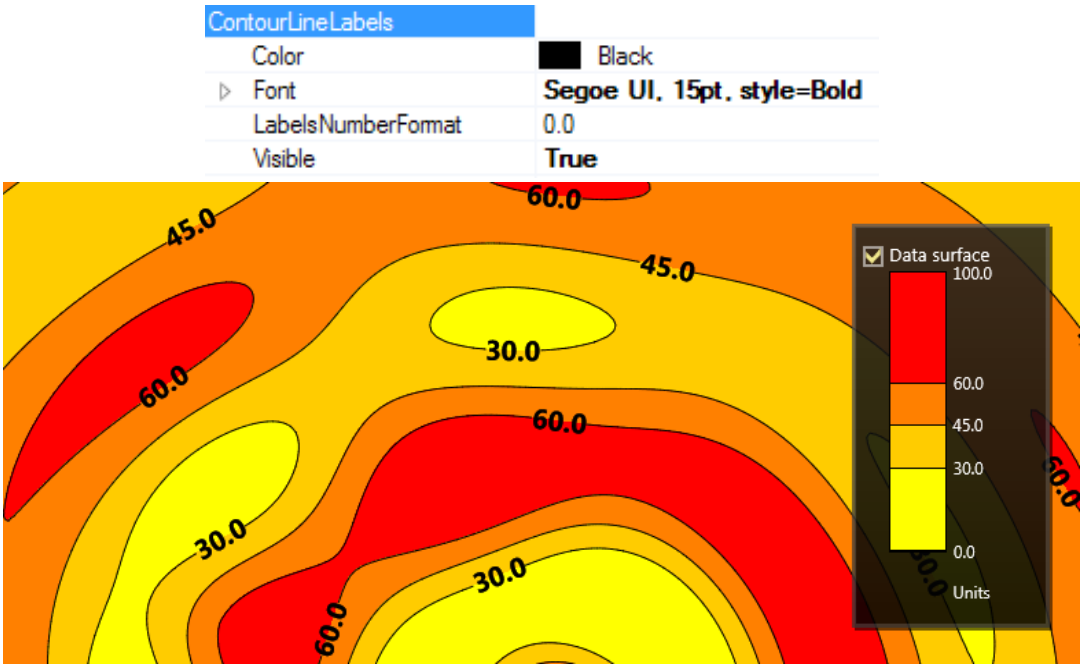


图 6-74. ContourLineLabels 属性及效果

用 ***LabelsNumberFormat*** 来自定义字符串格式化，例如设置小数位数。

6.23 IntensityMeshSeries

演示示例： Animated intensity mesh; Intensity mesh, static geometry; Intensity mesh, circle/polar geometry

IntensityMeshSeries 几乎与 ***IntensityGridSeries*** 类似。最大的不同点在于系列节点可以任意地放置于 X-Y 空间内。换句话说，这种系列并不呈现为矩形。线框线条可用 ***WireframeType*** 属性设置为可见。同时设置 ***ShowNodes*** 为 true 可显示出节点。

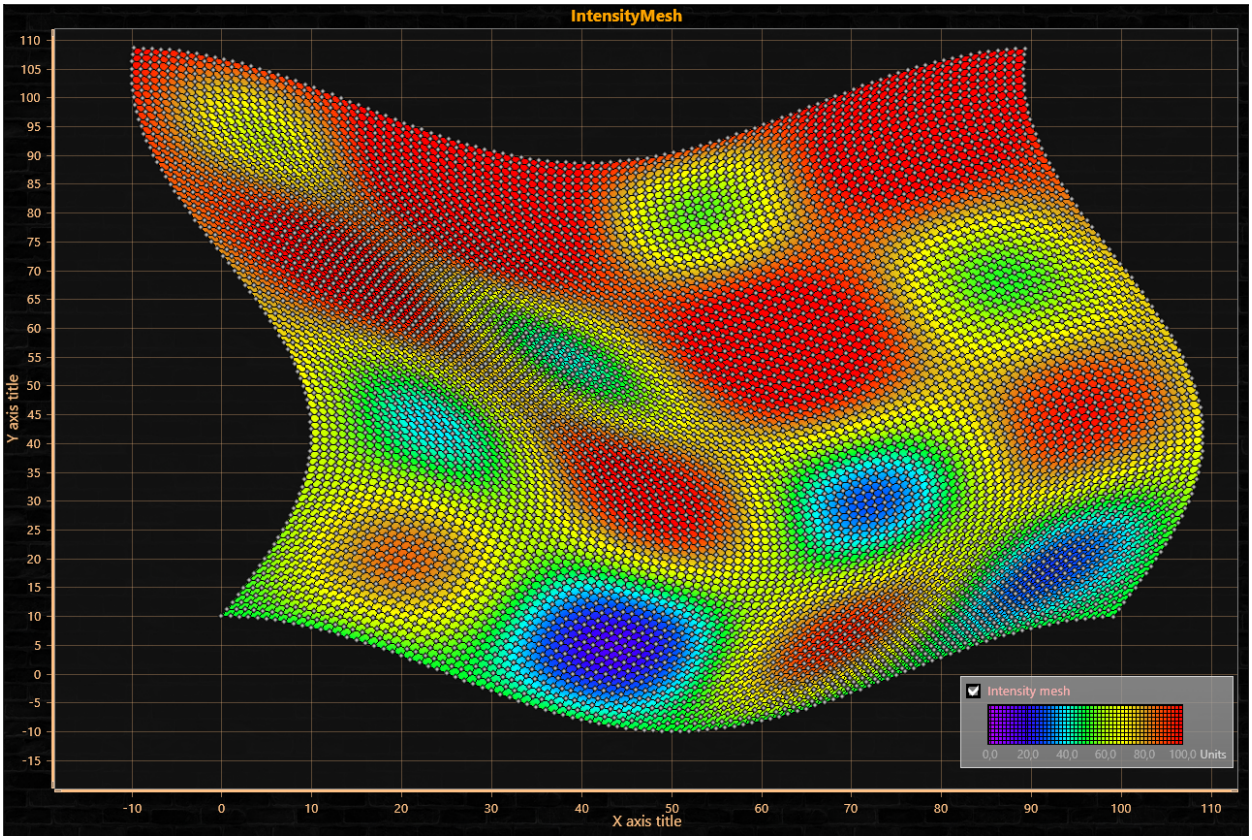


图 6-75. IntensityMeshSeries，任意定位每个节点的 X 和 Y 值。 WireframeType = Wireframe ， ShowNodes = true.

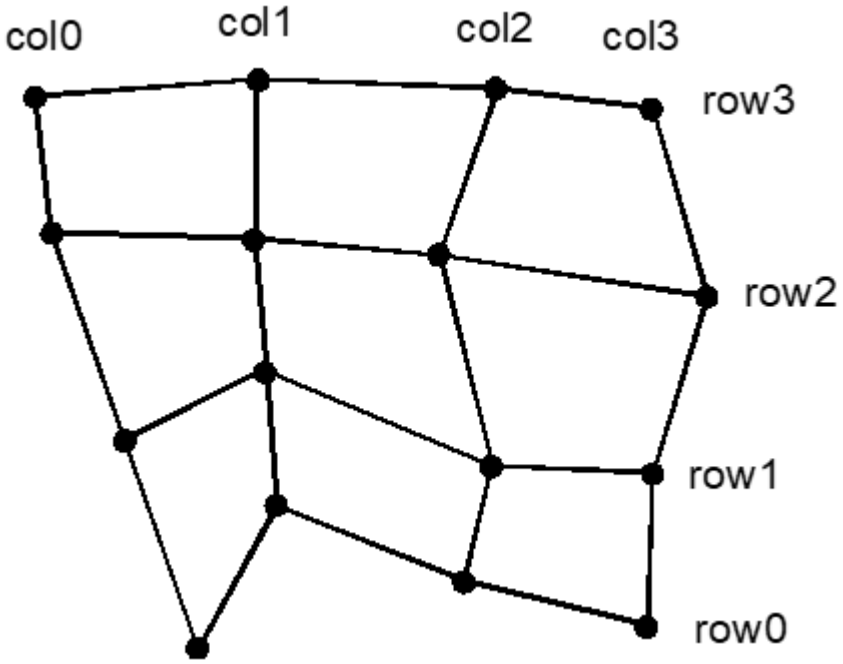


图 6-76. 强度网眼节点 SizeX = 4, SizeY =4.

6.23.1 几何形状变化时的强度网眼数据设置

当 ***X***、***Y*** 和 ***Value*** 字段同时更新时，需遵照以下指令说明操作。

- 设置 ***SizeX*** 和 ***SizeY*** 属性，为网眼赋予以列和行表示的尺寸。
- 为所有节点设置 X、Y 和 Value：

采用数据数组索引的方法

```
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（）; //通知新值已准备好并将刷新
```

采用 SetDataValue 的另一种方法

```
for （int nodeIndexX = 0; nodeIndexX < columnCount; nodeIndexX ++）
{
        for （int nodeIndexY = 0; nodeIndexY < rowCount; nodeIndexY ++）
        {
          meshSeries.SetDataValue（nodeIndexX, nodeIndexY,
                    xValue,
        yValue,
                    value,
                    Color.Green）; //本例中没有使用源点颜色，所以这里用什么颜色都行
        }
}
meshSeries.InvalidateData（）; //通知新值已准备好并将刷新
```

6.23.2 几何形状无变化时的强度网眼数据设置

当 ***Data*** 数组 ***IntensityPoint*** 结构的 ***Value*** 字段更新时，需按照这些指令说明操作。这是更新数据的性能优化方法，例如在热成像或环境数据监测解决方案中，每个节点的 ***X*** 和 ***Y*** 值保持在同一位置。

6.23.2.1 创建系列及其几何图形

- 设置 ***Optimization*** 为 ***DynamicValuesData***
- 设置 ***SizeX*** 和 ***SizeY*** 属性，为网眼赋予以列和行表示的尺寸。
- 为所有节点设置 X、Y 和 Value：

```
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（）; //根据节点重建几何形状并重新绘制
```

6.23.2.2 定期更新值

- 仅设置所有节点的值：

```
    for (int nodeIndexX = 0; nodeIndexX < columnCount; nodeIndexX ++)
    {
            for (int nodeIndexY = 0; nodeIndexY < rowCount; nodeIndexY ++)
        {
                meshSeries.Data[nodeIndexX, nodeIndexY].Value = value;
        }
    }
meshSeries.InvalidateValuesDataOnly ( ); //仅更新数据值
```

6.24 Bands（带）

演示示例：Bands; Statistic analytics; Long data analysis; Zoom bar chart

Bands 带可以看做是系列。它们具有与其他系列相同的用户界面操作，但是一个 band 系列只包含一个带。一条带是一片垂直或水平区域，从一侧图边距延伸到另一侧。用 ***Binding*** 属性可以将一条带连接到一条 Y 轴或 X 轴。如果将带连接到 Y 轴，必须还要设置 ***AssignYAxisIndex*** 属性。如果将带连接到 X 轴，则忽略 ***AssignYAxisIndex*** 属性，或将其设置为 **unassigned** （-1）。

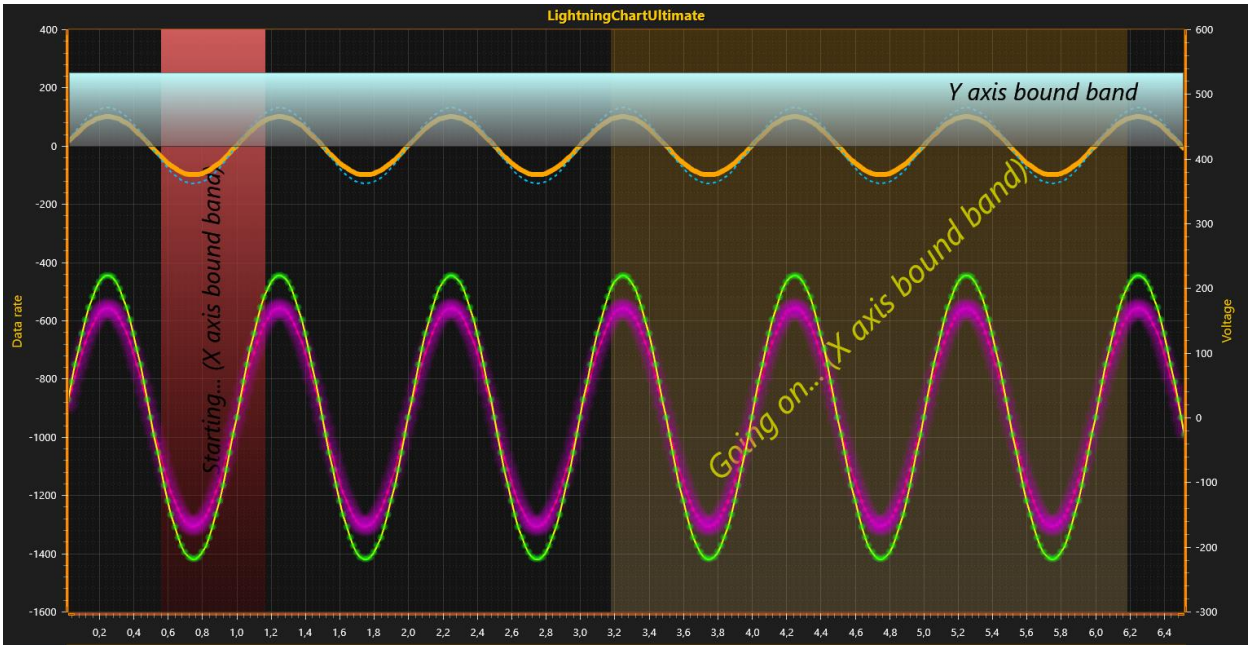


图 6-77. 几个带与线条系列

如果要把带置于线条或柱状系列后面，则设置 ***Behind*** 属性为 true 。用 ***ValueBegin*** 和 ***ValueEnd*** 属性可以设置带的边缘，即为连接的轴上的值。用鼠标可以将带拖动到其他位置。拖动边缘可以调整带的大小，即更新了被拖动边缘的值、 ***ValueBegin*** 或 ***ValueEnd***。

6.25 Constant lines （恒量线）

演示示例： Oscilloscope; Lissajous monitor; Signal reader; Areas; Segments with splitters

恒量线和带一样也可以当做是系列。恒量线与 Y 轴连接，呈现为一条水平线，自图表左边延伸至右边。通过 ***Value*** 属性可以设置水平高度。用鼠标可以垂直拖动恒量线。设置 ***Behind*** 属性为后，恒量线绘制于线条系列和柱状系列的后面，否则绘制在前面。

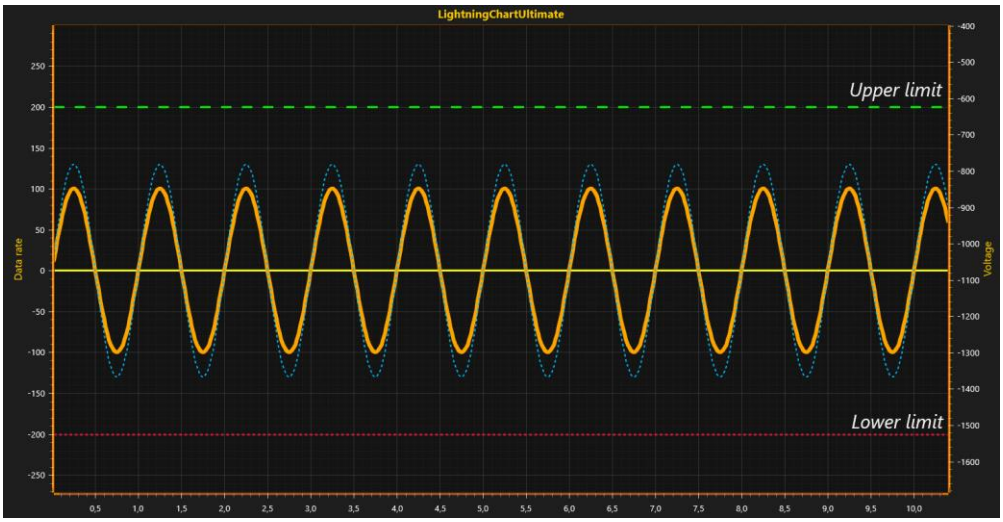


图 6-7850.恒量线分列于正弦系列两旁。

6.26 Annotations 注释

演示示例： Annotations; Custom rendering; Intensity grid mouse control; Multi-channel cursor tracking; Stocks and bars; Annotations table

注释可以在图表区域的任何地方显示鼠标交互的文本标签或图形。注释可以通过鼠标移动，调整大小，旋转，它们的对象和位置可以改变。或者可以通过代码来控制这些操作。如果必须在屏幕上渲染自定义图形，注释也很有用，因为它们可以以不同的样式和形状呈现。在 ***ViewXY.Annotations*** 集合中创建 ***AnnotationXY*** 对象。

当移动鼠标至注释上方时，可进入鼠标交互的编辑状态。此时可以对注释重定位，调整其大小，旋转，并指定箭头指向哪里。

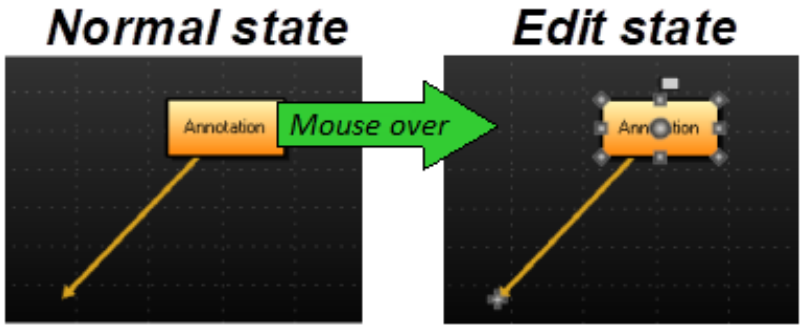


图 6-78. 移动鼠标至注释上方进入编辑状态；移走鼠标则退出编辑状态

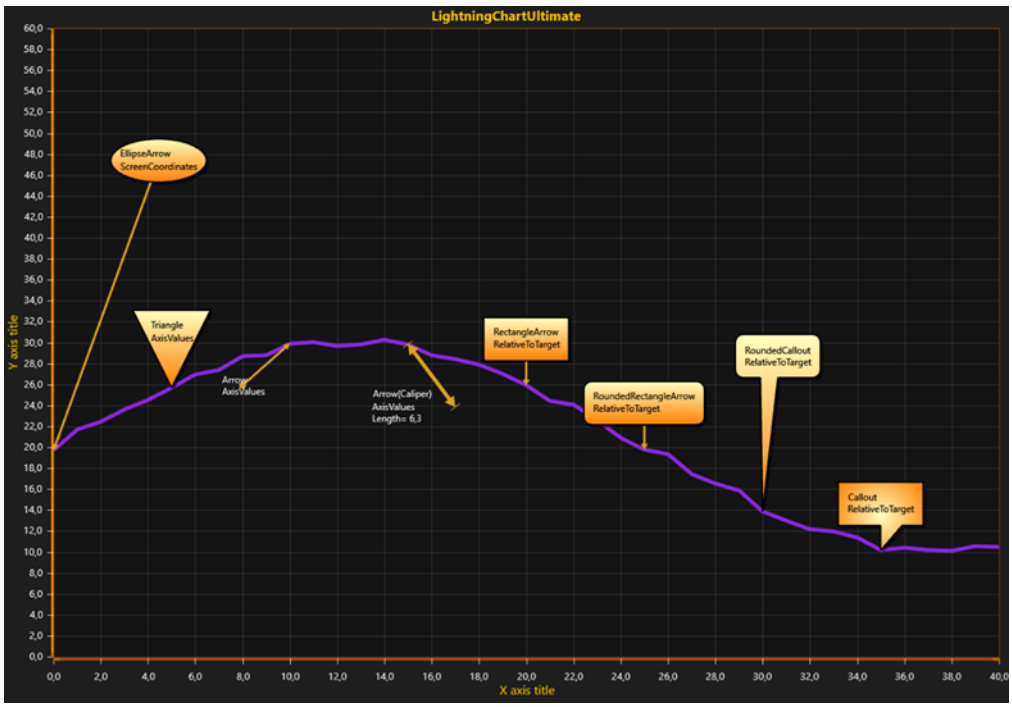


图 6-79. 各种样式的 AnnotationXY 对象，分置与一条线系列周围。用 *Style* 属性选择注释形状。

6.26.1 控制对象（Target）和位置（Location）

Target 为箭头终点所指，即箭头或标注尖端所指向的点。***Target*** 可以在轴值或在屏幕坐标中设置；具体可以用 ***TargetCoordinateSystem*** 在 ***AxisValues*** 和 ***ScreenCoordinates*** 二者中选择。如果选择了、***AxisValues***，通过 ***TargetAxisValues*** 属性设置箭头线指向的位置（箭头线的末端）；另外一个则是通过 ***TargetScreenCoords*** 在屏幕坐标中设置。

Location 是箭头的起始点，可以通过屏幕坐标、轴值或者与 ***Target*** 的相对偏移量来设置。用 ***LocationCoordinateSystem*** 在 ***LocationScreenCoords***、***LocationAxisValues*** 或 ***LocationRelativeOffset*** 三个方法中进行选择来控制位置。 ***Location*** 还是文本区域旋转的中心点。

Anchor 属性控制着文本区域在 ***Location*** 上的放置方式。设置 ***Anchor.X*** = 0.5 且 ***Anchor.Y*** = 0.5，箭头的起点居于中间；设置 ***Anchor.X*** 0.1 且 ***Anchor.Y*** = 0.25，箭头起点位于如下图所示的靠近左上角：

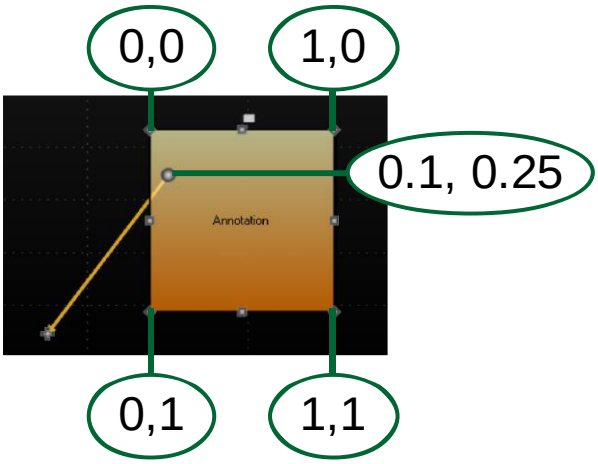


图 6-79. Anchor 值说明； Current Anchor.X = 0.1 且 Anchor.Y = 0.25. 当 anchor 值在 0...1 内，箭头起始点便位于文本区域内部。

6.26.2 用鼠标来移动、旋转及调整大小

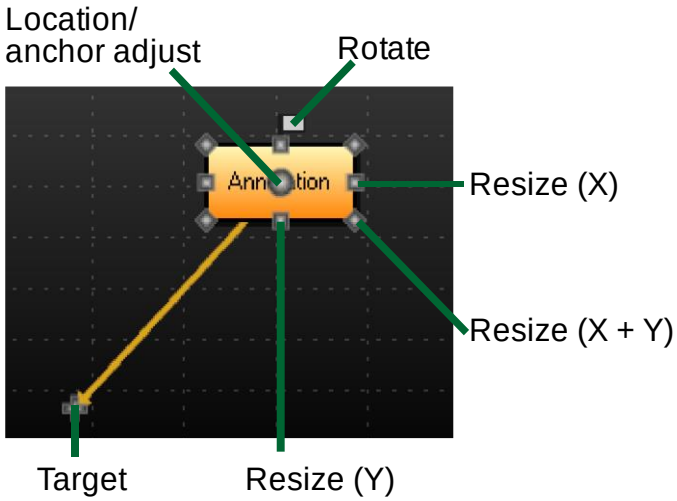


图 6-80. 注释的鼠标交互节点

从 ***Target*** 拖动可移动箭头的末端。从文本区域拖动可设置新的 ***Location***。通过拖动圆形的 location/锚节点，可以同时调整 ***Anchor*** 和 ***Location*** 属性，将文本框保持在相同的位置。

按住 ***Shift*** 键，然后拖动 resize （X）或 resize （Y）节点来调整大小, 采用对称操作，两边会同时调整。按住 ***Shift*** 键，然后从角位置上的节点 resize （X+Y）拖动，在调整大小时还会保留屏幕纵横比。在旋转操作时，***Shift*** 键可帮助以最近的 15 度倍数的旋转角度转动。

6.26.3 调整外观

通过设置 ***Style*** 属性可选择注释的外形；其中的选项有：***Rectangle***、***RectangleArrow***、***RoundedRectangle***、***RoundedRectangleArrow***、***Arrow***、***Callout***、***RoundedCallout***、***Ellipse***、***EllipseArrow***、***Triangle*** 和 ***TriangleArrow***。

如果样式中带有箭头，可使用 ***ArrowLineStyle***、***ArrowStyleBegin*** 和 ***ArrowStyleEnd*** 来控制箭头的图案。箭头末端的样式有以下几个选项： ***None***、***Square***、***Arrow***、***Circle*** 和 ***Caliper***。

用 ***Fill*** 可修改注释的填充效果。编辑状态的鼠标交互节点外观可以从 ***NibStyle*** 中进行更改。用 ***TextStyle*** 则可以控制文本区域内的字体设置与文本对齐情况。用 ***BorderLineStyle*** 和 ***CornerRoundRadius*** 可以控制边框线条的外观。

6.26.4 尺寸大小设置

用 ***Sizing*** 属性可控制注释文本框的大小：

- ***Automatic*** 根据内容来调整大小，并给边框留出 ***AutoSizePadding*** 空间。
- ***AxisValuesBoundaries*** 可以根据轴值来设置注释的尺寸，具体可以用 ***AxisValuesBoundaries.XMin***、***XMax***、***YMin*** 和 ***YMax*** 来定义。
- ***ScreenCoordinates*** 可以通过屏幕坐标来设置尺寸，具体可以使用到 ***SizeScreenCoords.Height*** 和 ***Width***。

6.26.5 保持文本区域可见

开启 ***KeepVisible*** 后，注释的文本区域被强制放在图形内部，这时用鼠标或通过代码移动注释时，都不会将其移动到图形外。在平移图形视图或调整轴时，将重新定位注释以显示在图形内部。

6.26.6 在轴上显示注释

通过设置 RenderBehindAxes = True，注释显示在轴下方。 所有裁剪和 Z 排序在这种情况下，功能不可行。 如果 ClipInsideGraph 设置为真，则 RenderBehindAxis 无效。

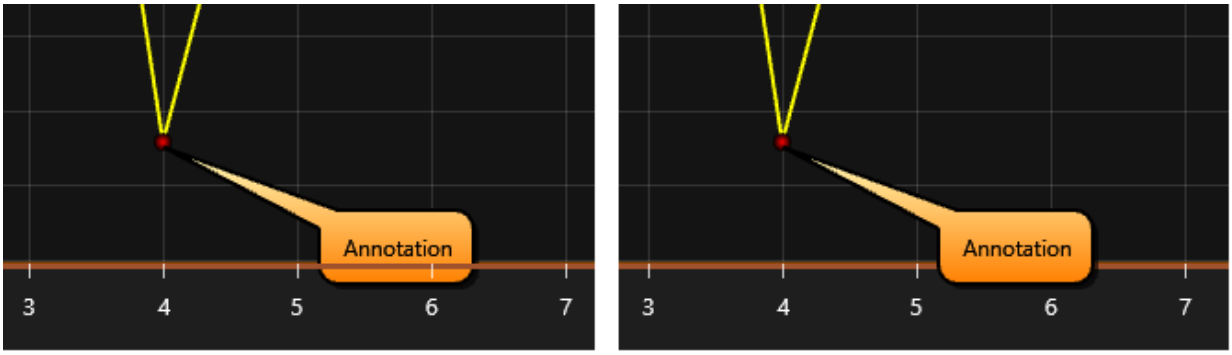


图 6-81. 左侧， RenderBehindAxes = True；右侧， RenderBehindAxes = False. 二者中的 ClipInsideGraph 都设置为 False

6.26.7 图形内剪辑

开启 ***ClipInsideGraph*** 后，可在图形内部对注释剪辑。禁用后，注释也可在图表的图边距区域渲染。

开启 ***ClipWhenSweeping*** 后，当设置 ***ScrollMode*** 为 ***Sweeping*** 模式时，注释不会在扫描间距区域显示。

6.26.8 控制 Z 序

当设置 ***Behind*** 属性为其默认值 ***False*** 时，注释显示在系列的上方。当设置为 ***True*** 时，注释在系列之前渲染，因此会出现在系列的下方。

注释按照它们在注释列表中的顺序显示，同时将 ***Behind*** 筛选程序作为主控制器。采用注释的 ***ChangeOrder*** 方法，例如鼠标事件处理程序，注释的 Z 序能够快速更改。用以改变 Z 序的选项有：

- ***BringToFront*** 注释处于最前
- ***SendToBack*** 置后
- ***MoveBack*** 向后移动一步
- ***MoveFront*** 向前移动一步

6.26.9 LayerGrouping 性能优化

当要显示数百个带有可见文本的注释时，文本渲染延迟便开始产生重要的影响。默认情况下，文本渲染按照 Z 序进行，让文本牢牢地保留在注释中。

设置 ***LayerGrouping = True***，可以提高性能，而且图表将只使用两个平面注释文本层。一个用于设置 ***Behind*** 为 ***True*** 的注释，另一个用于设置 ***Behind*** 为的 ***False*** 注释。这可以大大提高性能。另一方面，如果有注释与其他注释重叠，则文本将出现渲染错误。

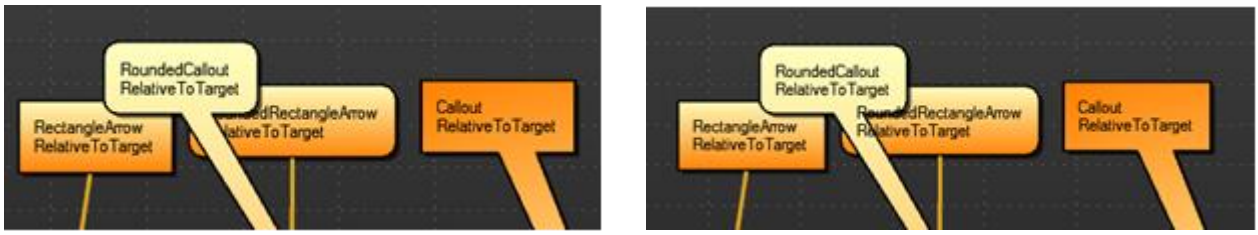


图 6-82. 左侧， LayerGrouping = False；右侧， LayerGrouping = True, 文本的 Z 序丢失。

当采用 Style = Arrow 或设置注释填充不可见时，通常不会出现 Z 序的限制。

6.26.10 轴的值与屏幕坐标之间的转换

在一些情况中， ***Location*** 或 ***Target*** 可能需要在混合配置中加以定义，如屏幕坐标中的为 X，轴的值为 Y，反之亦然。采用轴 ***ValueToCoord*** 的方法可以将一个轴的值转换为一个屏幕坐标，用 ***CoordToValue*** 可以将一个屏幕坐标转换为一个轴的值（如第 6.2.12 章节所述）。

6.27 图例框

演示示例： Multiple legends; Heatmap legends; Segments with splitters

从第 v.8 版本开始，ViewXY 支持在同一个张图形中出现多个图例框。在 ***ViewXY.LegendBoxes*** 集中插入这些图例框。

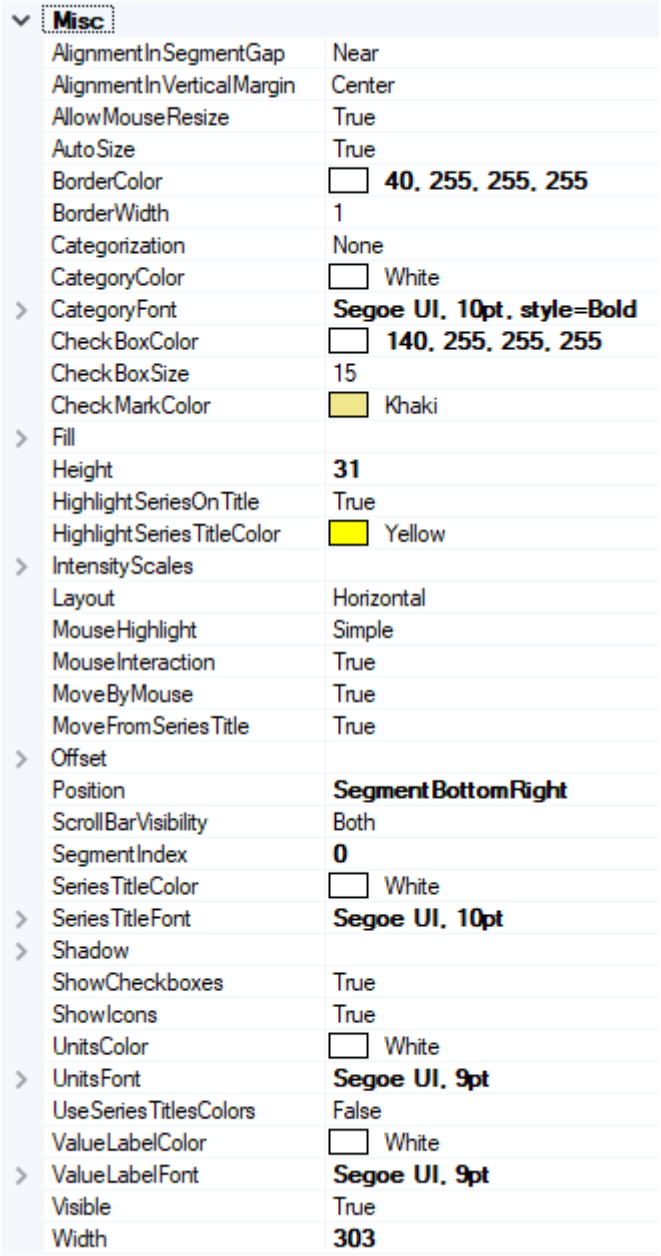


图 6-83. 庞大的 LegendBoxXY 属性树

6.27.1 在图例框中设置隐藏/显示某序列图

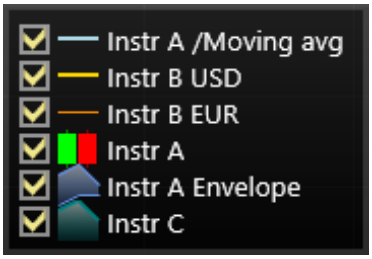


图 6-84. 在图例框中显示有系列名和图标。取消选定某个系列的勾选框可以将其隐藏。

6.27.2 在图例框中显示系列

默认情况下，所有系列都显示在图例框中。 如果不应列出特定系列，请为该系列设置 **series.ShowInLegendBox = False**。

如果使用多个图例框，请使用 **series.LegendBoxIndex** 选择首选图例框。 系列只能出现在一个图例框中。 所有系列的默认索引为 0，这意味着除非另有说明，否则它们都将出现在同一个图例框中。

6.27.3 选择在哪个图形段中显示图例框

用 ***SegmentIndex*** 来控制在哪个段中显示图例框。只适用于基于段的 ***Position*** 选项。

6.27.4 修改复选框

若要显示或隐藏图例框中的复选框，请使用 ShowCheckBoxes 属性。CheckBoxColor 和 CheckMarkColor 可用于在 CheckBoxSize 控件时更改复选框的外观框的大小（以像素为单位）

```
_chart.ViewXY.LegendBoxes[0].ShowCheckboxes = true;
_chart.ViewXY.LegendBoxes[0].CheckBoxColor = Colors.Green;
_chart.ViewXY.LegendBoxes[0].CheckMarkColor = Colors.Blue;
_chart.ViewXY.LegendBoxes[0].CheckBoxSize = 15;
```

6.27.5 隐藏图标

设置 ***ShowIcons = False***，可隐藏图标。

6.27.6 修改强度系列的调色板刻度

要隐藏一个 ***IntensityGrid*** 或 ***IntensityMesh*** 的调色板刻度，可设置 ***IntensityScales.Visible = False***。设置 ***ScaleSizeDim1*** 和 ***ScaleSizeDim2*** 属性，则可对其大小进行调整。刻度的边框以及名称的位置也可以修改。

图 6-85. 在底部的图片中设置 LegendBox.IntensityScales.Visible = false

6.27.7 控制位置

图例框可以自动放置或手动放置。自动放置可以让它们对齐到图形段的左侧/顶部/右侧/底部，或在图边距上。用 ***Position*** 属性可以控制位置。位置选项有：***TopCenter***、***TopLeft***、***TopRight***、***LeftCenter***、 ***RightCenter***、 ***BottomLeft***、 ***BottomCenter***、 ***BottomRight***、 ***Manual***。

如果视图分为数个片段，图例框可以根据其所属的段进行对齐（此可用 ***SegmentIndex*** 来控制）。对于基于图段进行控制，有以下几个选项可用：***SegmentTopLeft***、***SegmentTopCenter***, ***SegmentTopRight***、***SegmentBottomLeft***、***SegmentBottomCenter***、***SegmentBottomRight***、***SegmentLeftMarginCenter***、***SegmentRightMarginCenter***。

Offset 属性根据给定的数量从由 ***Position*** 属性决定的地方来移动位置。

```
// 设置图例框位置，偏移量从RightCenter位置移动
chart.ViewXY.LegendBoxes[0].Position = LegendBoxPositionXY.RightCenter;
chart.ViewXY.LegendBoxes[0].Offset = new PointIntXY（-15, -70）;
```

Manual 定位计算从图例框左上角到视图左上角的偏移量。注意，这与 ***TopLeft*** 选项不同，***TopLeft*** 选项是从图形区域的顶部计算的。

注意，当移动或改变图例框的大小时，其 ***Position*** 设置为 ***Manual***，并更新 ***Offset*** 属性以显示新的位置。

如果不设置 ***Position*** 回除‘***Manual***’之外的选项，自动图例框对齐是禁用的。由于在 ***Position*** 选项之间切换时 ***Offset*** 未更新，图例框有时看起来会消失（位于视图之外）。这个问题可通过将 ***Offset*** 设置回 0,0 来修正。

6.27.8 为图形段间的图例框分配空间

设置 ***ViewXY.AutoSpaceLegendBoxes = True*** 后，将对段之间的额外空间进行分配，以适当安置其中的图例框。

注意，段间也要分配 ***ViewXY.AxisLayout.SegmentsGap***。

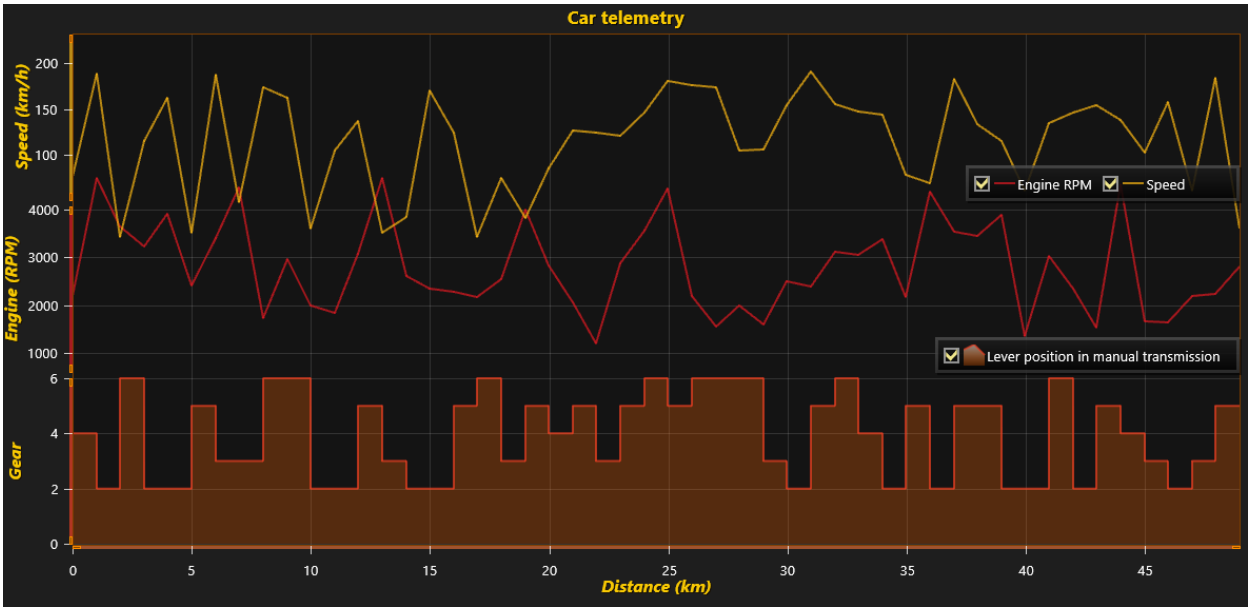


图 6-86. Position = SegmentBottomRight. AutoSpaceLegendBoxes = False.

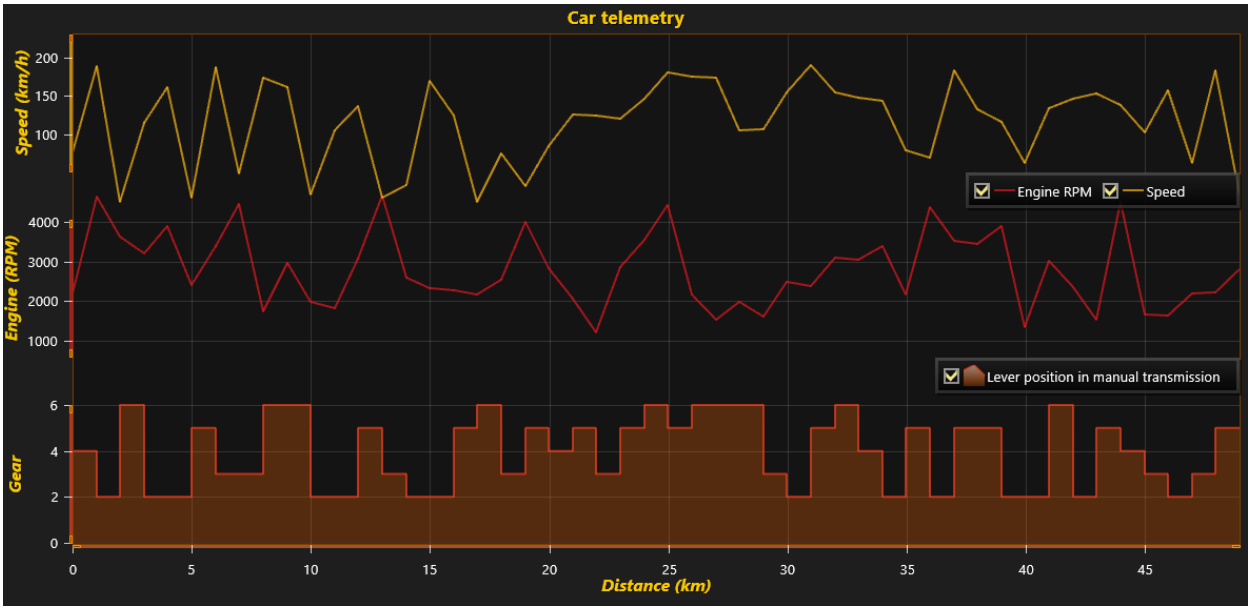


图 6-87. Position = SegmentBottomRight. AutoSpaceLegendBoxes = True.

6.27.9 段间距内的图例框对齐

要将图例框垂直对齐到指定段附近，可设置 ***AlignmentInSegmentGap = Near***。要将其垂直对齐到段与段之间间距的中心，可设置 ***AlignmentInSegmentGap = Center***。

6.27.10 在同一图边距中的几个图例框水平对齐

AlignmentInVerticalMargin 属性具有 ***Left/Center/Right*** 几个选项。该属性控制将图例框（设置在同一个垂直图边距内）的水平位置。设置为相同的垂直边框。

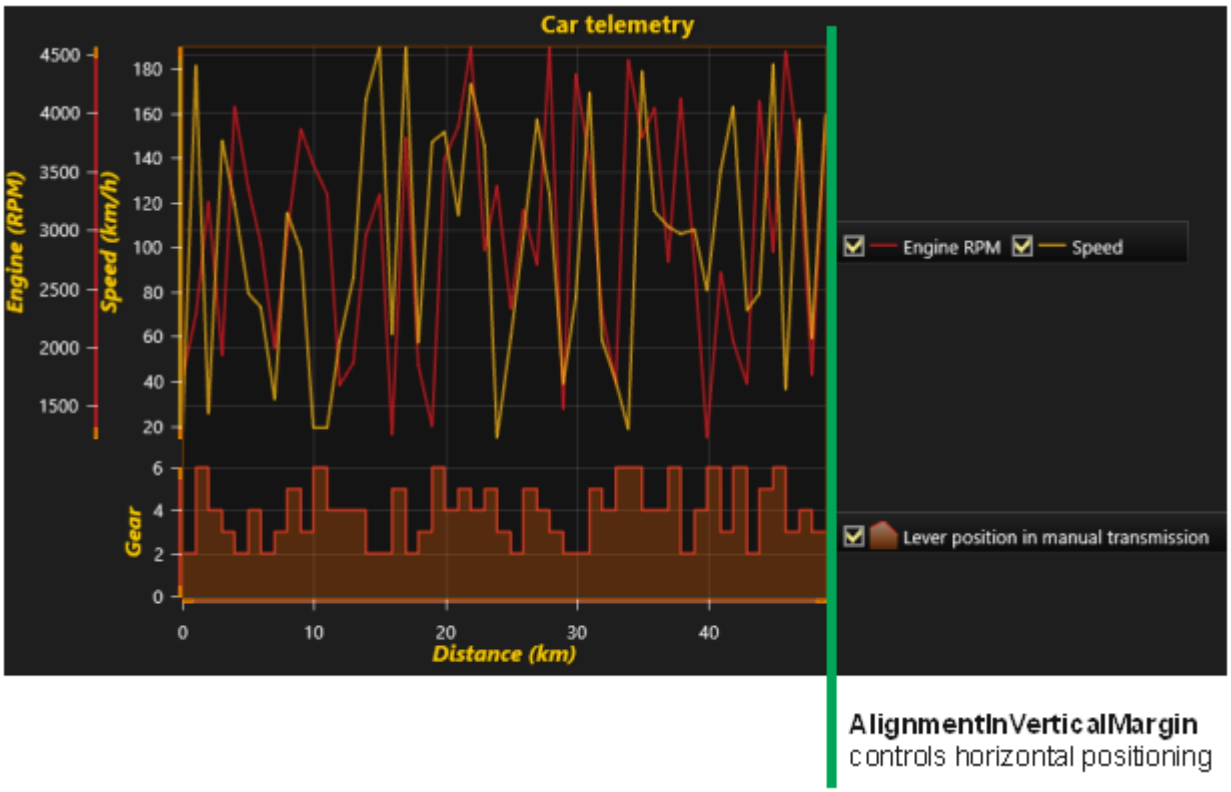


图 6-88. AlignmentInVerticalMargin = Left set for both Legend boxes.

6.27.11 图例框的大小调整与移动

图例框支持调整大小和滚动条。从边缘处抓取可调整大小。

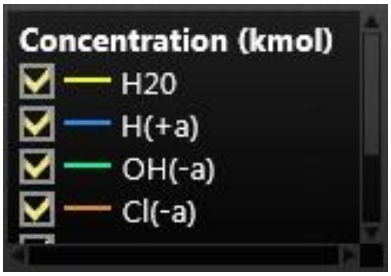


图 6-88. 图例框中的滚动体

注意，当移动或调整图例框大小时，其 ***Position*** 设置为 ***Manual***， 并更新 ***Offset*** 属性，以显示新的位置（参阅第 6.22.8 章）。

6.27.12 图例框事件

除了典型的鼠标点击事件之外，图例框还有几个特定的事件。

- **CheckBoxStateChanged** 在系列复选框的状态从选中变为未选中时触发，反之亦然。 该事件具有 IsChecked 属性，以获取当前复选框的状态和 Series 属性以检查受影响的系列。

- **SeriesTitleMouseClick、SeriesTitleMouseDown** 等是特殊事件，仅当与图例框中的系列标题交互时才会触发。 如果类似的事件，例如 MouseClick，用于图例框和系列标题，系列标题事件将优先。 MouseOverOn 和 MouseOverOff 也有 Series 属性来检查哪个系列受到影响。请注意，上述事件仅在禁用 MoveFromSeriesTitle 属性时有效

```
// Using Legend box events.
_chart.ViewXY.LegendBoxes[0].MoveFromSeriesTitle = false;
_chart.ViewXY.LegendBoxes[0].CheckBoxStateChanged +=
Legend_CheckBoxStateChanged;

private void LegendBox_CheckBoxStateChanged(object sender,
Arction.Wpf.Charting.Views.CheckBoxStateChangedEventArgs e)
{
    if (e.Series is PointLineSeries series) // Get the affected series.
    {
        series.LineStyle.Color = Colors.Yellow;
    }
}
```

6.28 缩放与平移

使用 ***ZoomPanOptions*** 来控制缩放与平移设置。

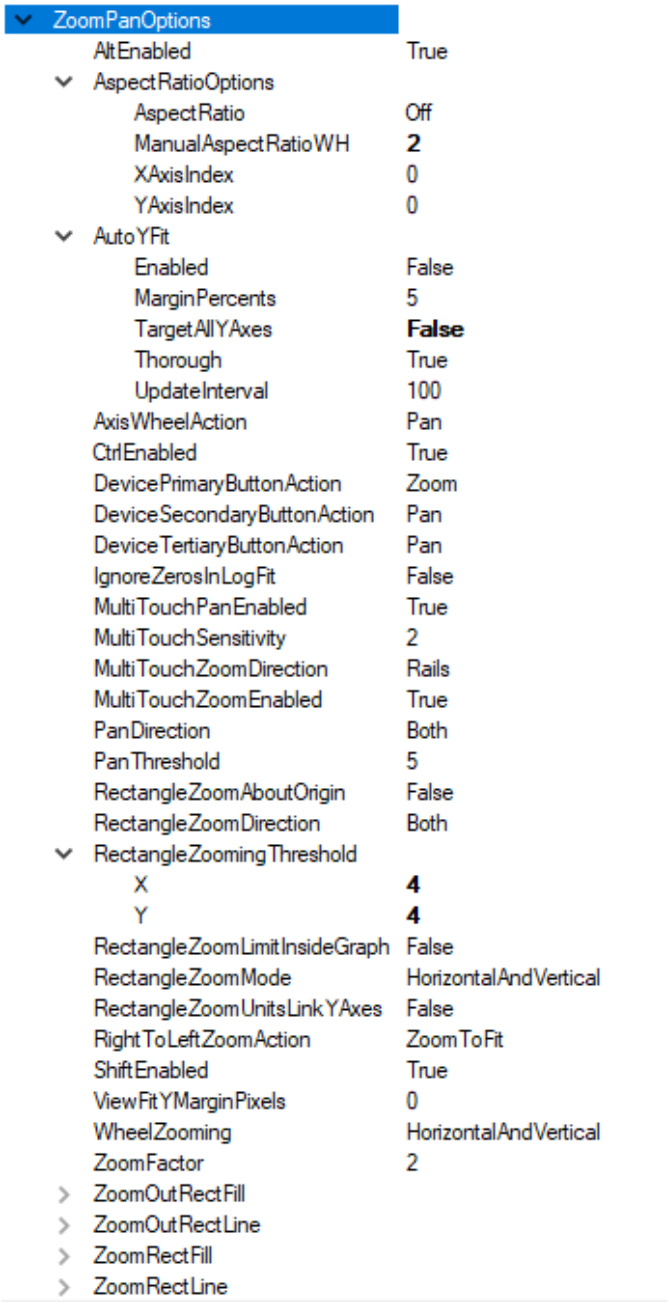


图 6-89. ZoomPanOptions 属性及子属性

缩放和平移是可配置的，可以通过鼠标左键或右键来实现。缩放也可以用鼠标滚轮来实现。

6.28.1 触控屏幕进行缩放

将两根手指放在图表上，并紧手指可缩小，或分离手指可放大。

图表会试着检测是否视图进行水平或垂直缩放，或两者同时进行。这一功能称作 'zooming with rails'，可由 ***MultiTouchZoomDirection*** （***Free/XAxis/YAxis/Rails***）控制。

通过在 X 轴或 Y 轴或其标签上捏并/展开手指，可只对该特定的轴缩放。

设置 ***MultiTouchZoomingEnabled = false*** 可以禁用触摸缩放。

6.28.2 触控屏幕进行平移

将两根手指放在屏幕上，以同等的速度滑动两根手指来平移视图。

一些系统支持惯性平移，也因此可以将手指从屏幕上"甩"离，视图会继续平移，然后慢下来，最后停下。

通过把手指放在 X 轴或 Y 轴或其标签上，然后滑动手指，可只对该特定的轴平移。

6.28.3 鼠标左键操作

设置 ***LeftMouseButtonAction*** 为 ***Zoom***，可开启用鼠标左键缩放。设置其为 ***Pan***，可开启平移。要禁用从鼠标左键来缩放与平移，可设置其为 ***None***。

6.28.4 鼠标右键操作

设置 ***RightMouseButtonAction*** 为 ***Zoom***，可开启用鼠标右键缩放。设置其为 ***Pan***，可开启平移。要禁用从鼠标右键来缩放与平移，可设置其为 ***None***。

6.28.5 RightToLeftZoomAction

当 ***DevicePrimaryButtonAction*** 或 ***DeviceSecondaryButtonAction*** 设置为 ***Zoom*** 时，可应用 ***RightToLeftZoomAction***。***RightToLeftZoomAction*** 规定了当从右向左进行鼠标缩放时会有什么操作效果。（鼠标的 X 按下坐标>完成坐标）.

可用选择如下：

ZoomToFit: 适应所有的 Y 轴和 X 轴，使全部所属的系列数据都显示出来。使用 ***ViewFitYMarginPixels*** 设置值大于 0，对轴进行缩放，使得在 Y 轴的最小和最大端的给定像素空间都保留空数据。

RectangleZoomIn: 矩形缩放，与从左到右缩放类似。

ZoomOut：用 ***ZoomFactor*** 属性缩小。

RevertAxisRanges: 设置轴值为具体数值，这些值在视图经过缩放或轴范围修改后恢复。在每条轴上的 ***RangeRevertEnabled*** 属性，控制着是否应该恢复轴的范围。若开启该属性后，当从右向左拖动鼠标，然后释放鼠标键，***RangeRevertMinimum*** 和 ***RangeRevertMaximum*** 属性则会应用到轴上。

PopFromZoomStack: 设置上次缩放时使用的轴范围，也就是返回到先前的缩放级别。

6.28.6 使用鼠标键进行缩放

6.28.6.1 单击鼠标进行变焦缩放

用 ***ZoomFactor*** 属性可控制变焦的近/远程度。若要应用负变焦效果，则将值设为倒数值（1/倍数）。使用鼠标光标位置作为变焦中心点来应用变焦缩放。

X 向缩放：

光标聚焦在图表控件时，按下 Shift 键。Zoom X 光标出现。单击已配置的鼠标键以放大，单击另一个键以缩小。

Y 向缩放：

光标聚焦在图表控件时，按下 Ctrl 键。Zoom Y 光标出现。单击已配置的鼠标键以放大，单击另一个键以缩小。当使用堆叠的 ***YAxisLayout*** 时，缩放适用于所有图形段（Y 轴）。按下 Ctrl 和 Alt 键，Y 向缩放仅应用于鼠标点击过的图形段。

同时按下 Shift 和 Ctrl（+Alt）二键，可对 X 和 Y 向应用缩放。

6.28.6.2 通过鼠标指针操作缩放

使用配置的鼠标按钮，在要缩放的区域周围拖动一个矩形，从左上角到右下角。X 和 Y 维度都会受到影响。要限制图形区域内的矩形绘制/缩放，请启用 RectangleZoomLimitInsideGraph 属性。

通过 **RectangleZoomMode** 属性可以对缩放矩形所绘成的位置进行配置，以及应该如何处理不同的方向。默认情况下，该属性设置为 **HorizontalAndVertical**, 也就是将对通过从左向右拖动鼠标所绘成的区域进行缩放。相对地，从右向左拖拽鼠标绘成的是缩小矩形。 如果 **RectangleZoomMode** 设置为 **Horizontal** 或 **Vertical**, 只有这一个方向会缩放。

X 轴和 Y 轴都具有 ***ZoomOrigin*** 属性，可以用于设置缩放矩形的中心位置。如果 If ***RectangleZoomMode*** 设置为 ***AboutYAxisZoomOrigin***、***AboutYAxisZoomOrigin*** 或 ***AboutXYZoomOrigin***，the position set via 通过 ***ZoomOrigin*** 设置 的位置将总是用作缩放矩形的中心点。

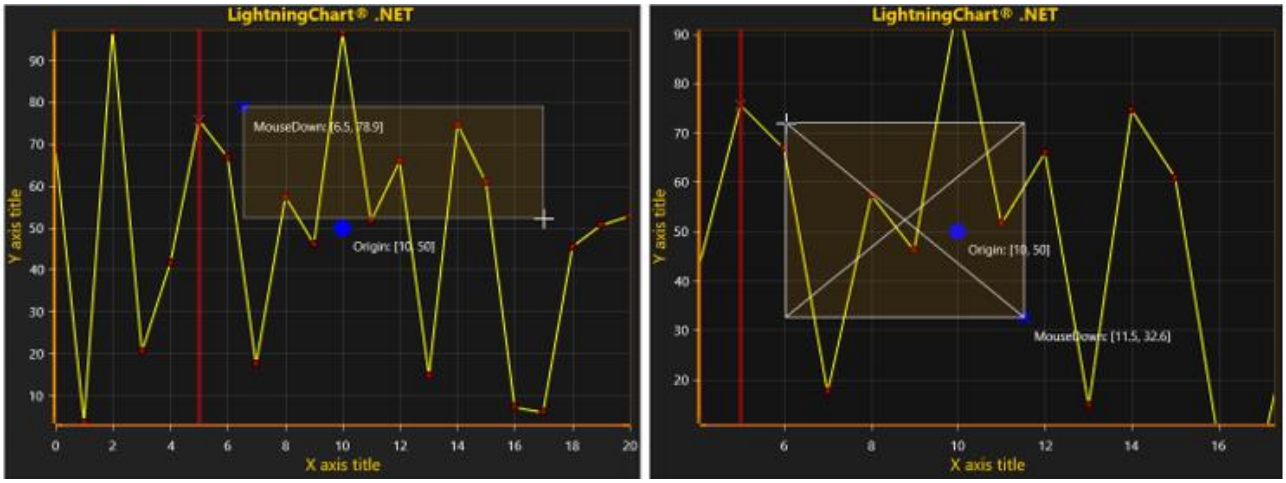
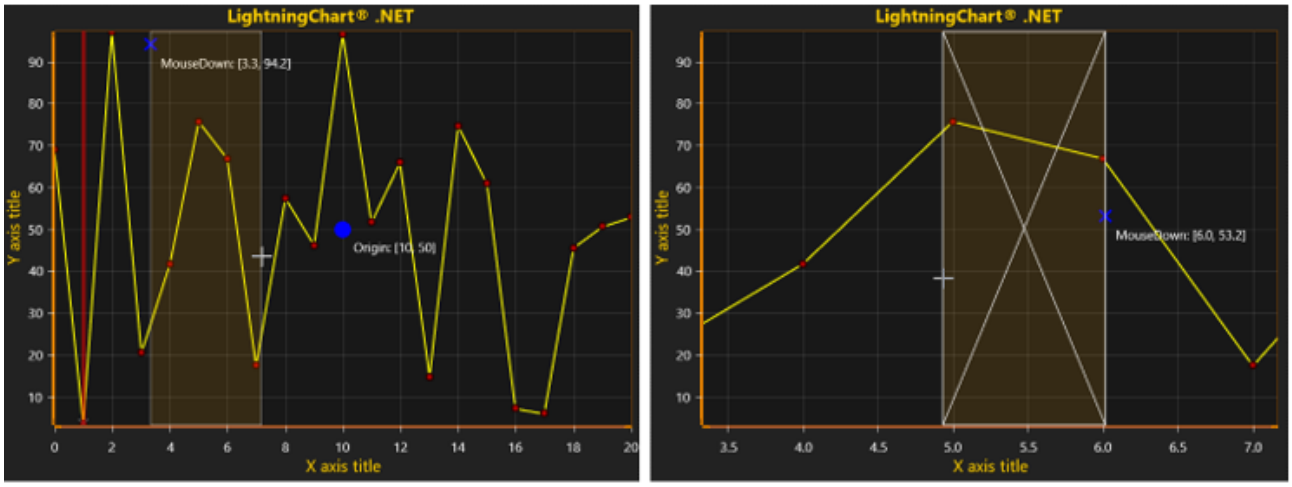


图 6-90. 标准设置： ZoomPanOptions.RectangleZoomMode = HorizontalAndVertical. ZoomOrigin 对缩放矩形没有影响。



使用配置的鼠标按钮，在要缩放的区域周围拖动一个矩形，从左上角到右下角。X 和 Y 维度都会受到影响。要限制图形区域内的矩形绘制/缩放，请启用 RectangleZoomLimitInsideGraph 属性。

图 6-91. ZoomPanOptions.RectangleZoomMode = Horizontal. 只对 X 轴缩放，Y 轴保持不变。

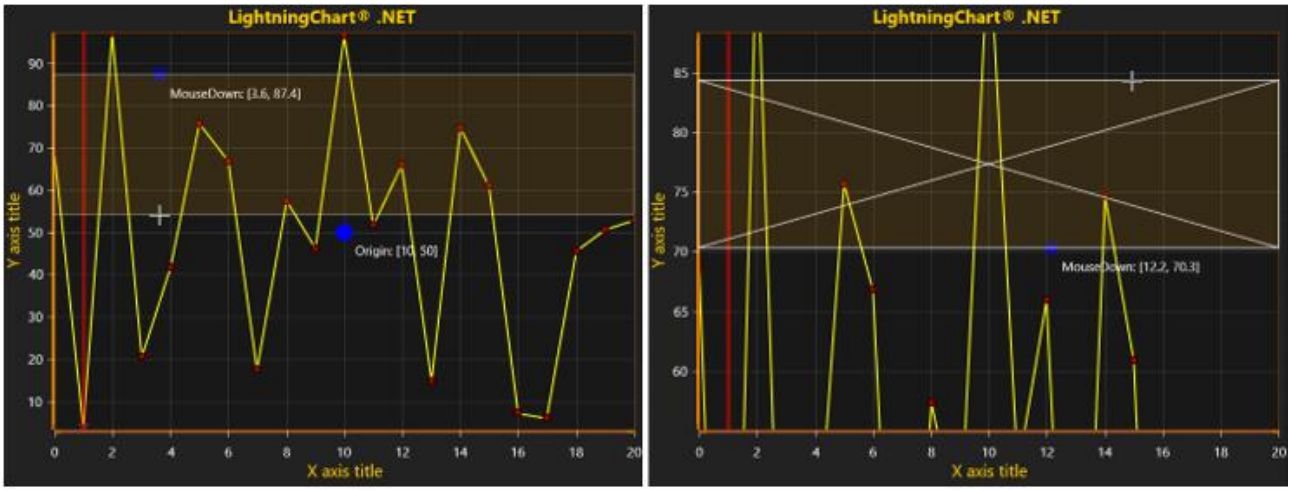


图 6-92. ZoomPanOptions.RectangleZoomMode = Vertical. 只对 Y 轴缩放，X 轴保持不变。

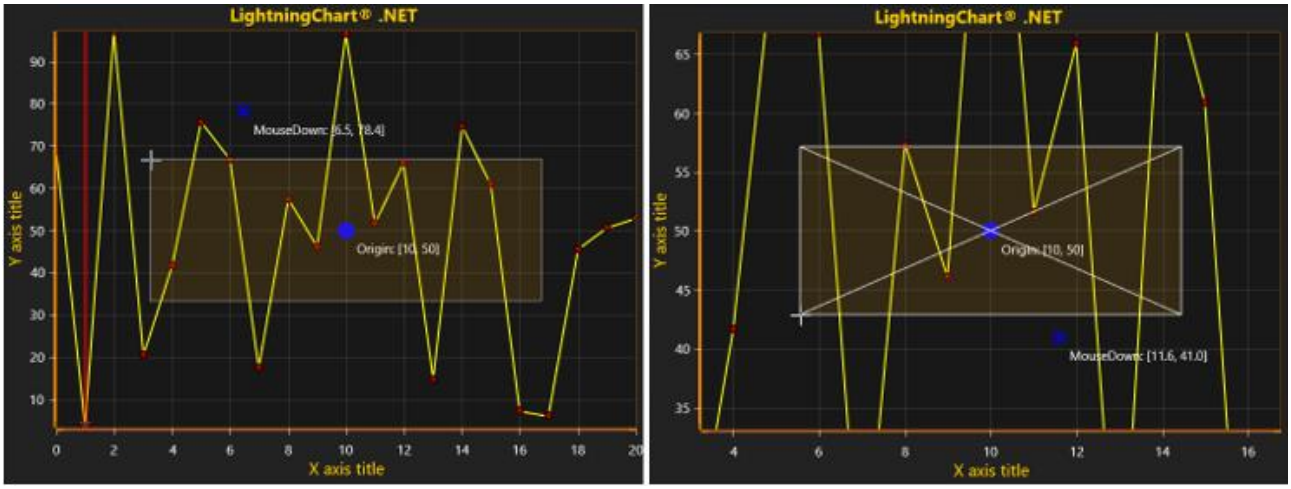


图 6-93. ZoomPanOptions.RectangleZoomMode = AboutXYZoomOrigin.不用管鼠标下移位置。缩放矩形与 ZoomOrigin（两个轴）和鼠标当前位置成比例绘制。

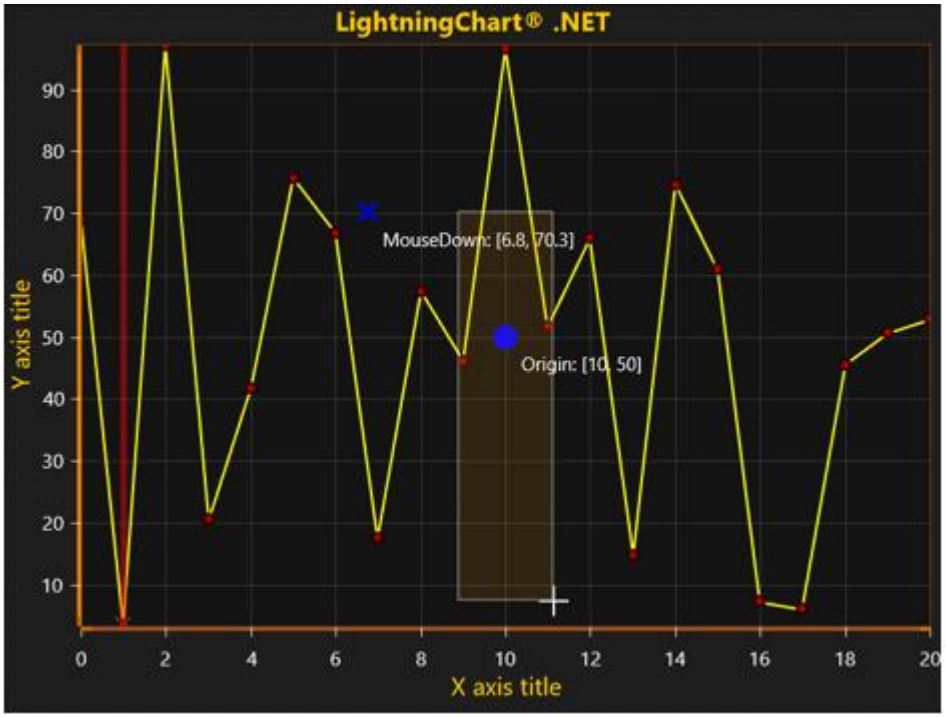


图 6-94. ZoomPanOptions.RectangleZoomMode = AboutXAxisZoomOrigin. 缩放矩形以 XAxis.ValueOrigin 为中心。鼠标向上和下移决定了缩放矩形的垂直高度。

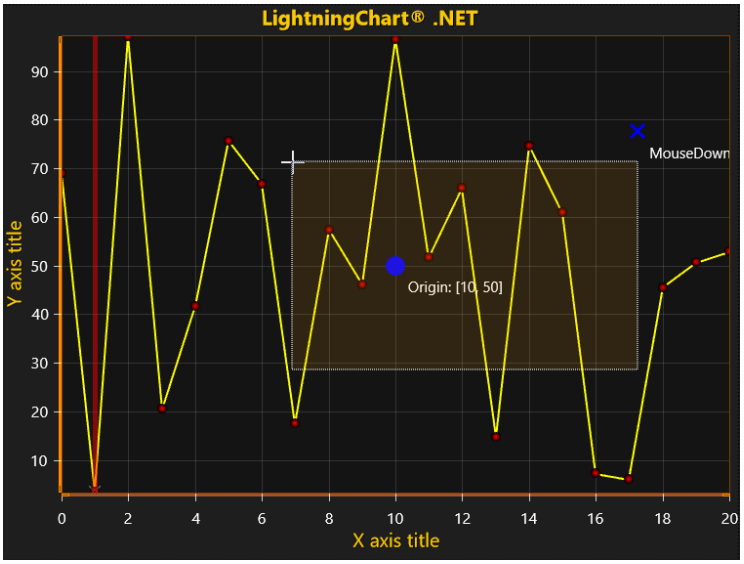


图 6-95. ZoomPanOptions.RectangleZoomMode = AboutYAxisZoomOrigin. 缩放矩形以 YAxis.ValueOrigin 为中心。鼠标向上和下移决定了缩放矩形的水平宽度。

6.28.6.3 配置矩形放大

当 DevicePrimaryButtonAction 或 DeviceSecondaryButtonAction 属性设置为 Zoom 时，缩放时会出现放大矩形。可以使用 ZoomRectFill 和 ZoomRectLine 属性修改缩放矩形边框和填充样式。

6.28.6.4 配置缩小矩形

当 RightToLeftZoomAction 设置为 ***ZoomToFit***、***ZoomOut***,、***RevertAxisRanges*** 或 ***PopFromZoomStack*** 后，缩放时会出现缩小矩形。用 ***ZoomOutRecFill*** 可配置其填充效果，用 ***ZoomOutRectLine*** 可配置线条样式。

6.28.7 用鼠标滚轮进行缩放

启用滚轮缩放后，向上滚动鼠标滚轮可放大，向下滚动鼠标滚轮可缩小。缩放中心是鼠标光标的位置。使用 ZoomFactor 调整鼠标滚轮缩放强度（注意，ZoomFactor 是间接使用的，因为还考虑了滚轮的增量）。按住 Shift 键，缩放仅适用于 X 维度。按住 Ctrl 键，缩放仅适用于 Y 维度。

6.28.8 使用设备滚轮在轴上的缩放和平移

用 ***AxisWheelAction*** 可配置应用于轴上的鼠标滚轮操作的效果。

None: 鼠标滚轮什么都不做

Zoom: 只缩放光标指针下方的轴

Pan: 只平移光标指针下方的轴

ZoomAll: 如果光标指针在一条 X 轴上，则缩放所有 X 轴，或者在一条 Y 轴上时，缩放所有 Y 轴。仅当设置 ***YAxisLayout*** = ***Layered*** 后，则应用与其他轴。

PanAll: 如果光标指针在一条 X 轴上，则平移所有 X 轴，或者在一条 Y 轴上时，平移所有 Y 轴。仅当设置 ***YAxisLayout = Layered*** 后，则应用与其他轴。

6.28.9 用鼠标键平移

配置 ***DevicePrimaryButtonAction*** 或 ***DeviceSecondaryButtonAction*** 为 ***Pan*** 来进行平移。按下已配置的鼠标键，拖动图形区域。要停止平移，释放鼠标键即可。若设置 ***PanDirection*** 为 ***Both***，根据拖动量，通过平移操作可滚动 X 和 Y 轴。设置 ***PanDirection Vertical***，则只针对 Y 轴；而 ***PanDirection Horizontal*** 则只以 X 轴为对象。在平移产生作用之前，用 ***PanThreshold*** 设定一些像素容差。使用为图表控件分配的 ContextMenuStrip 控件非常方便，可以防止每次平移停止时它都打开。

6.28.10 Ctrl、Shift 与 Alt 键的启用/禁用

缩放操作支持这些修改键，而且在认情况下，它们是处于启用状态的。若要禁用这些键，可设置 ***AltEnabled = False***、 ***CtrlEnabled = False*** 或 ***ShiftEnabled = False***。

6.28.11 使用代码进行缩放

用 ***ZoomByFactor（…）*** 方法以一个中心点及一个缩放倍数来缩放。用 ***Zoom（…）*** 方法以矩形来缩放。 ***ZoomToFit（）*** 方法适合调用“缩放以适应”操作（适合所有的 Y 轴和 X 轴，以显示所有的系列数据）。

6.28.12 使用代码对轴进行缩放

为 X 或 Y 轴的 ***Minimum*** 和 ***Maximum*** 属性设置值。用 ***SetRange（…）*** 可对二者同时设置。

6.28.13 围绕可配置的原点进行矩形缩放

开启 **RectangleZoomUnitsLinkYAxes** 后，所有具有同样的 ***Units.Text*** 字符串的 Y 轴获得与经过矩形缩放的轴一样的 Y 轴值域。

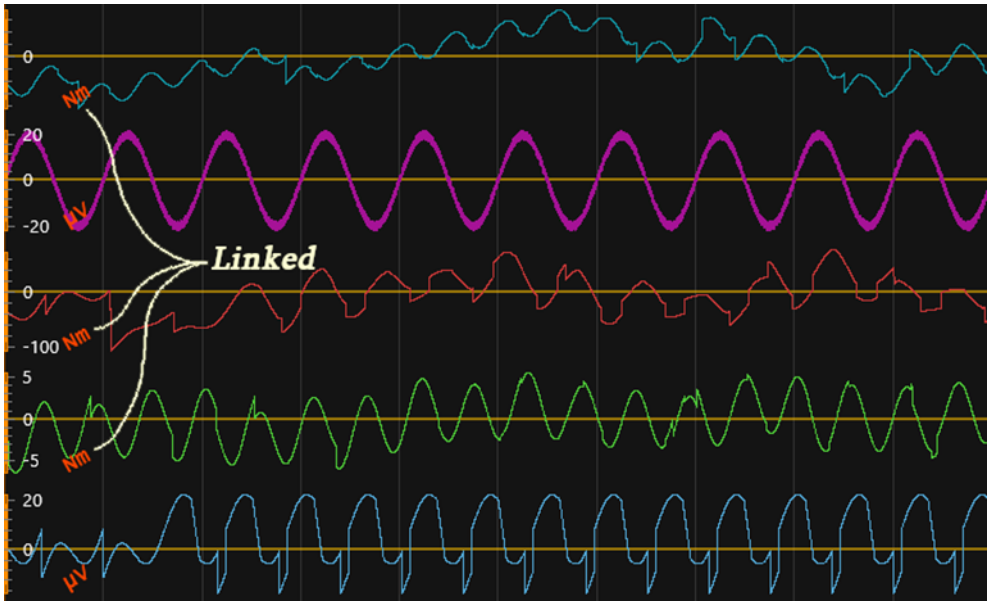


图 6-93. 5 个 Y 轴堆叠的视图。当对一个图形段应用矩形缩放时，图形段的 Y 轴被缩放，新的 Y 轴值域则会复制到所有具有相同 Units.Text 的 Y 轴。

6.28.14 自动 Y 轴适应

演示示例： Signal reader; Audio L+R, area, spectrograms; Waveform, persistent spectrum

用 ***AutoYFit*** 属性可控制自动的 Y 轴调整。自动 Y 轴适应可用于调整 Y 轴值域，以可见的 X 轴范围显示图表中的所有数据。这主要用于实时监控。每隔一段时间进行一次适应，用 ***UpdateInterval*** 可以毫秒为单位设置间隔。***MarginPercents*** 可用于定义系列和图形边界之间是否应该留有空白。启用 ***Through***, 可对所有数据进行适应分析，但可能会在性能关键的系统中引起一些消耗。若禁用，仅一小部分最新数据用于适应程序，并可能在某些应用程序中引发错误的反应。

AutoYFit 可通过 ***ZoomPanOptions*** 启用:

```
_chart.ViewXY.ZoomPanOptions.AutoYFit.Enabled = true;
```

哪些 y 轴是自动适应的也应该要定义。用 ***TargetAllYAxes***, 自动 Y 轴适应可同时应用与所有 Y 轴。相反，启用 ***AllowAutoYFit*** 可对每条 Y 轴分别应用。

```
// 对所有 Y 轴启用 AutoYFit
_chart.ViewXY.ZoomPanOptions.AutoYFit.TargetAllYAxes = true;

// 仅对该 Y 轴启用 AutoYFit
_chart.ViewXY.YAxes[0].AllowAutoYFit = true;
```

注意！ ***AxisY*** 类还有 ***Fit***（）方法可以 Y 轴方向进行适应。

6.28.15 Aspect ratio （屏幕高宽比）

AspectRatioOptions.AspectRatio 控制着 X/Y 比（或者地图上的经/纬比）。

默认情况下，将其设置为 ***Off*** 可以对 X 轴和 Y 轴的值域分别进行设置。设置屏幕高宽比为 ***Manual*** 后，可用 ***ManualAspectRatioWH*** 属性设置首选比率。通过改变 ***ManualAspectRatioWH***，调整 X 轴的 ***Minimum*** 和 ***Maximum*** 属性，来获得所需的高宽比。缩放操作将依据长宽比设置运行。

ManualAspectRatioWH 计算方法如下：

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

例如：

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

视图的宽与高取决于窗口大小。轴值域即为最小值到最大值。

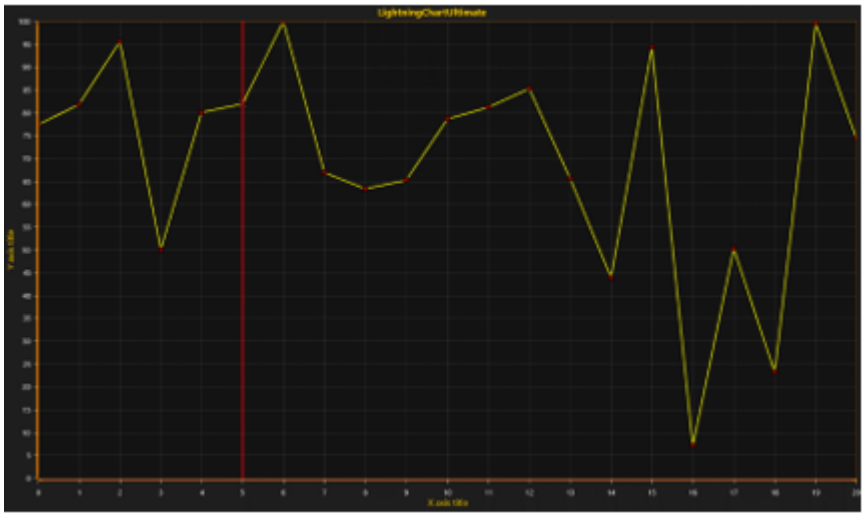


图 6-94. 图表的视图区域。其像素尺寸用于计算 ManualAspectRatioWH.

当 ***AspectRatio*** 设置的不是 ***Off*** 时， 轴刻度的尖不可用。

对于地图来说（参与第 6.31 章节），***AspectRatio*** = ***AutoLatitude*** 是以非常有用的选项。在不同位置查看地图时，***AutoLatitude*** 会动态改变高宽比。高宽比由视图的中心点决定。

6.28.16 从缩放与平移操作中除去特定的 X 轴或 Y 轴

- 要从缩放操作中排除掉特定的 X 轴或 Y 轴，设置

  axis.ZoomingEnabled = False

- 要从平移操作中排除掉特定的 X 轴或 Y 轴，设置

  axis.PanningEnabled = False

6.29 通过 NaN 值或其他值实现 DataBreaking

演示示例： Data breaking in series

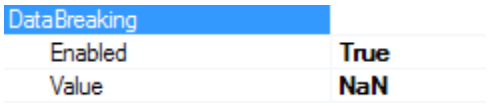


图 6-95. 在系列中支持 DataBreaking 的几个选项

以下系列类型支持数据中断：

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

LightningChart 跳过与指定的断开值匹配的数据点的渲染，但可正常渲染其他值。

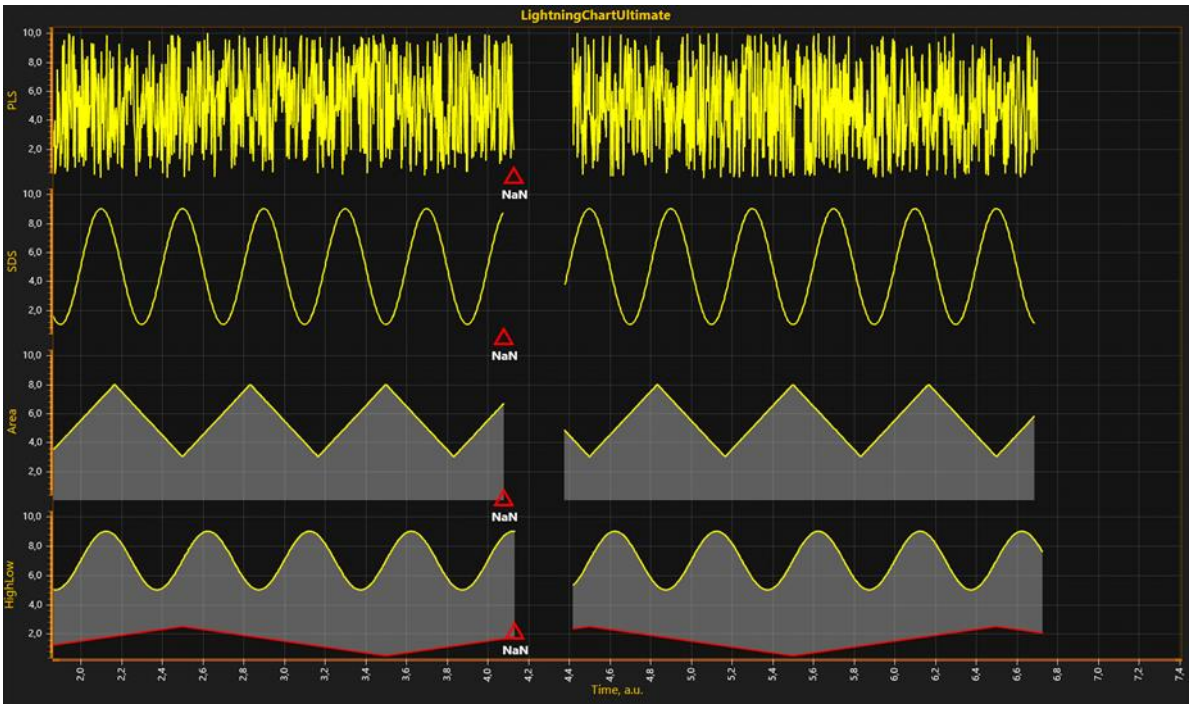


图 6-96. DataBreaking 用于 PointLineSeries、SampleDataSeries、AreaSeries 和 HighLowSeries.

注意! 当设置 DataBreaking.Enabled = True, 会引发明显的额外消耗，不建议用于需要非常高的实时数据速率的解决方案。可考虑使用 ClipAreas, 参阅第 6.30 章节。

例如，使用 NaN 中断 PointLineSeries 数据：

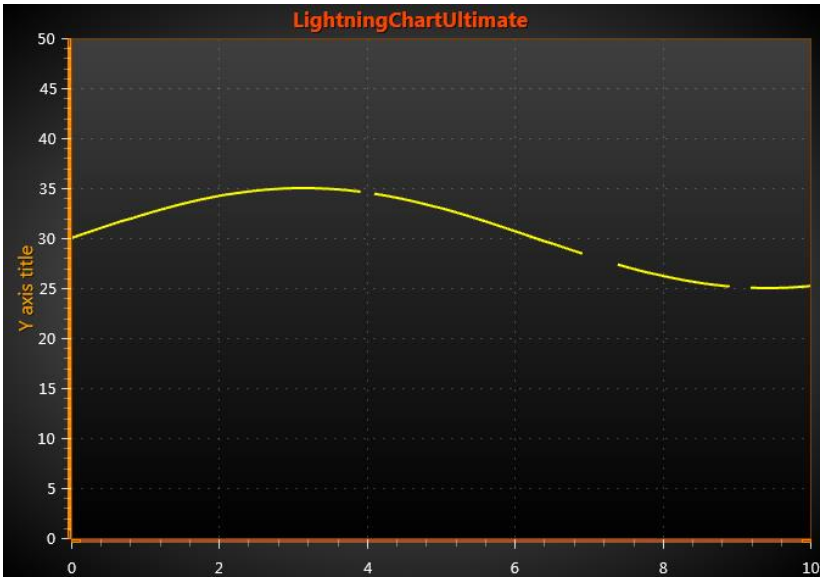


图 6-97. 使用 NaN 值中断 PointLineSeries.

代码：

```
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）;
}

//在Y数组中添加一些NaN值来标记中断点
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;

//设置DataBreaking Enabled来添加新系列
        PointLineSeries pls = new PointLineSeries（_chart.ViewXY, _chart.ViewXY.XAxes[0],
        _chart.ViewXY.YAxes[0]）;
pls.DataBreaking.Enabled = true;

//设置定义值的数据间隙（默认值= 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];
}

//为点线系列分配数据
pls.Points = points;
```

```
//将创建的点线系列添加到PointLineSeries列表中
_chart.ViewXY.PointLineSeries.Add（pls）;
```

6.30 ClipAreas

演示示例： Clip areas

与 ***DataBreaking*** 类似（参阅第 6.29 章节），***ClipAreas*** 可用于避免渲染一部分系列数据。这可以用来筛选不良数据范围，Y 值范围外的数据等等。

ViewXY 视图的系列具有 ***SetClipAreas*** 方法可设置或更新裁剪区域。它可接受一个 **ClipArea** 结构的数组。ClipAreas 数组可以频繁变更，*并且性能保持良好*，最多可高达数千个 ClipAreas。

ClipArea 适用于其被分配给的系列。注意此为处于渲染阶段的裁剪，当鼠标置于 ClipArea 上方时，如果 ClipArea 下有实际的数据，鼠标操作则会相应系列。

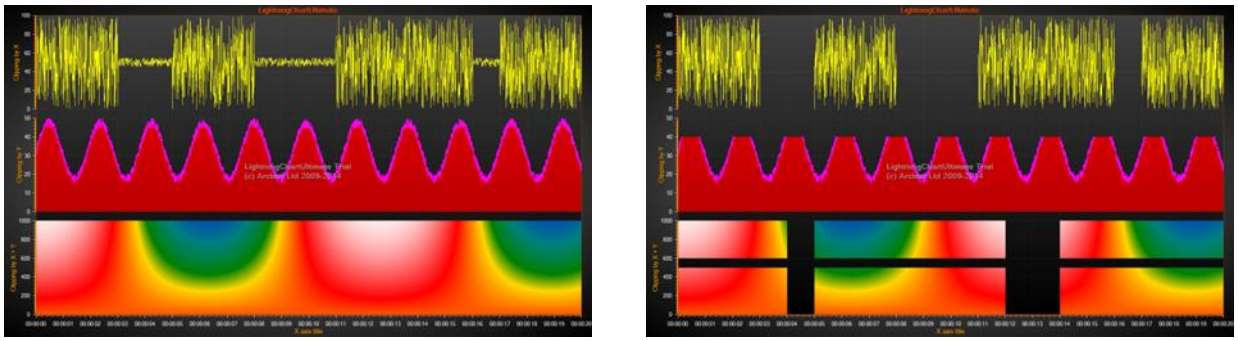


图 6-98. 为 2 个系列定义 ClipAreas； 分别为 PointLineSeries、AreaSeries 和 IntensityGridSeries. 左侧，未使用 ClipAreas ；右侧，启用 ClipAreas；对于黄色的 PointLineSeries, 经定义了 X 维度的裁剪区域来掩盖低振幅数据；对于红色的 AreaSeries, Y 维度的 ClipArea 从顶部剪切太高的振幅数据；对于 IntensityGridSeries, X 维度和 Y 维度的 ClipAreas 用于防止系列在特定区域中渲染。

使用 ClipAreas 是在性能方面应考虑的首选方法，它将线分割成多个数据段，而不是使用 DataBreaking 特性，或者在实时监视期间生成数百个单独的系列。

6.31 Maps 地图

使用 ***Maps*** 属性及其子属性，可显示地理地图。LightningChart 地图包括两个不同的种类：***vector maps*** 和 ***tile maps***。这些地图是以所谓的 *equirectangular* 投影来显示的。

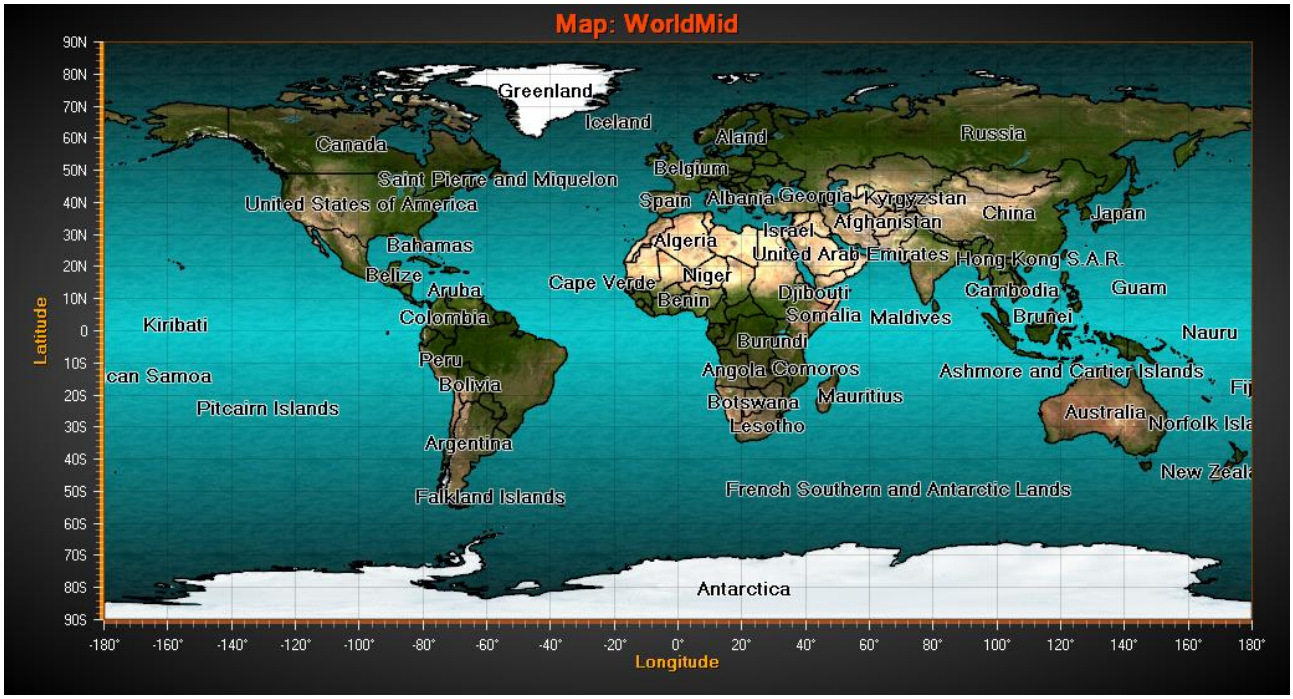


图 6-99. 世界地图的 Equirectangular 投影。 X 值域为从-180 到 180 度（180W 到 180E）， Y 值域为从-90 到 90 度 （90S 到 90N）。在此投影中，极地地区被拉伸地非常大。

该投影可使用 LightningChart *的系列类型和其他对象*，这些对象实际上都绑定到 X 轴和 Y 轴上，同时也绑定到地图上。

6.32 矢量地图

演示示例： World map; Map route; Map with environmental data; Wind data

地理矢量数据以***.md*** 扩展名存储于 LightningChart 地图文件中。LightningChart 以地图文件集传送。

X 轴表示经度，Y 轴表示纬度。显示地图坐标轴可参阅第 6.2.3 章节所示。地图坐标为十进制度数，纬度起点于赤道，经度起点于英国格林威治。

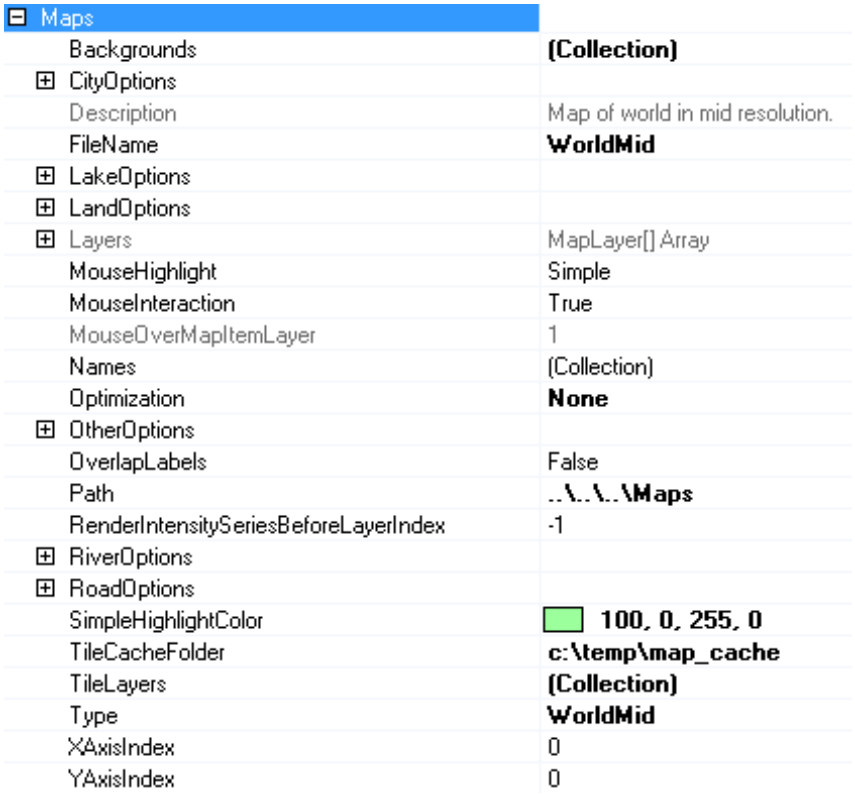


图 6-100. 地图的属性与子属性。该属性树用于矢量地图，除了 TileLayers 集和 TileCacheFolder 是用于瓦片地图。

6.32.1 选择有效的地图

为 ***Path*** 属性设置包含地图文件的目录名称。对于使用 LightningChart 传送的地图，用 ***Type*** 属性可以选择有效的地图。要使用自己的地图文件，可设置 ***FileName*** 属性。

如果不需要地图，可设置 ***Type*** 为 ***Off***。

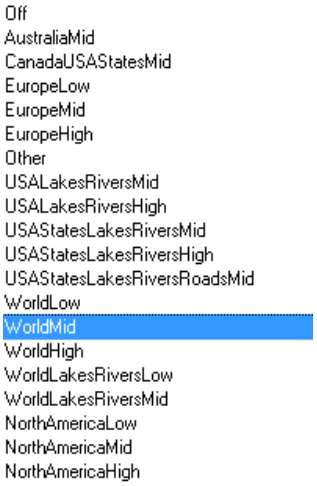


图 6-101. 地图类型选项。显示以 LightningChart 传送的地图。通过类型名称后缀可以分辨地图的大致精细级别。

一般而言，LightningChart 地图均制作的非常精细。对于实时监控解决方案来说，选择一份提供准确细节和性能级别的地图非常重要。

6.32.2 Aspect ratio（屏幕高宽比）

ViewXY.ZoomPanOptions.AspectRatioOptions.AspectRatio 控制 X/Y 比（或经/纬比）。

将其设置为 Off，以分别启用 X 轴和 Y 轴值域设置，可拉伸地图。在不同位置查看地图时，***AutoLatitude*** 会动态改变高宽比。高宽比由视图的中心点决定。设置高宽比为 ***Manual***，用 ***ManualAspectRatioWH*** 属性设置首选比。关于如何计算高宽比的详细说明可参阅第 6.22.16 章节。

6.32.3 图层及其外观设置

每个地图文件可包含多个图层。例如，陆地区域、湖泊、河流、道路和城市图层。从 ***Layers*** 数组属性可访问图层及其数据。

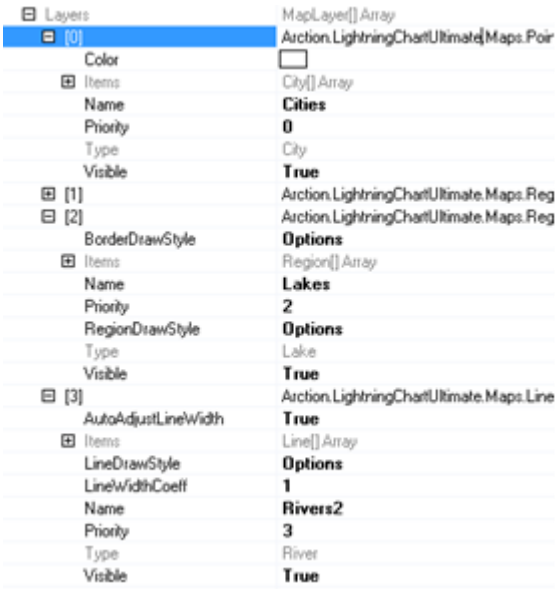


图 6-102. 在属性编辑器中打开地图图层详情

每个图层都具有一个特定的类型。图层外观选项可以用相应的选项属性来变更。用 ***LandOptions*** 可修改陆地区域的外观，用 ***LakeOptions*** 修改湖泊，用 ***RiverOptions*** 修改河流，用 ***RoadOptions*** 修改道路，用 ***CityOptions*** 修改城市，用 ***OtherOptions*** 修改未指明的图层类型。

图 6-103. 默认的 LandOptions，及对应的欧洲视图

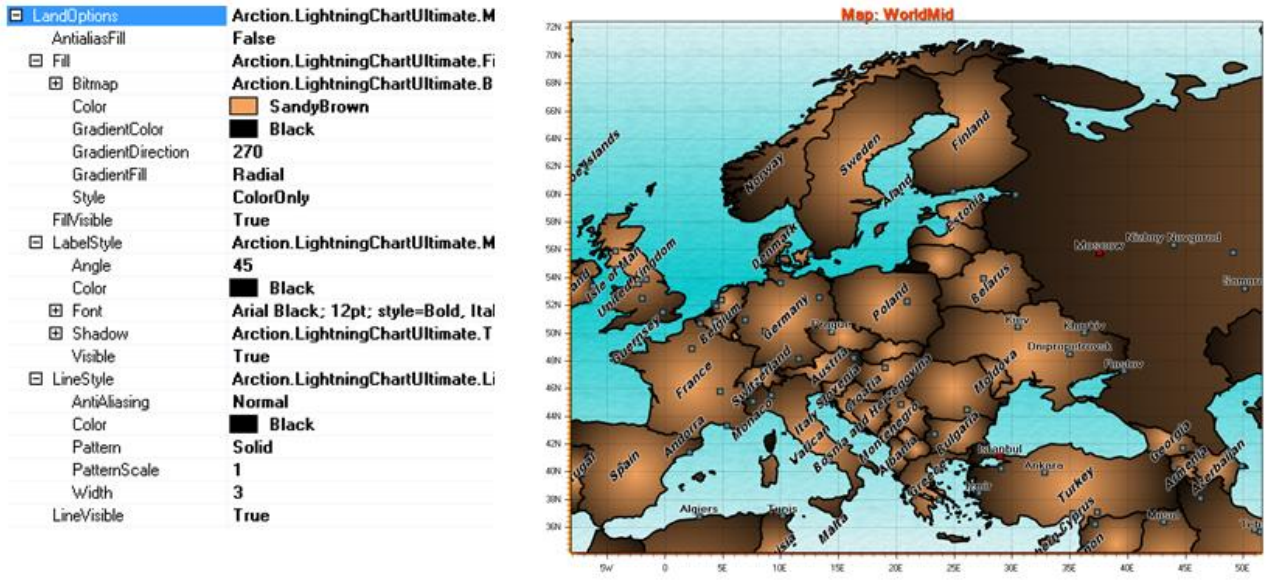


图 6-104 修改后的 LandOptions.

5.25.3.1 为每一图层项设置单独的填充和边框样式

每个地图元素填充或边框外观可以单独设置。更改 ***BorderDrawStyle*** 和 ***RegionDrawStyle*** 属性为 ***Individual*** 。然后访问 Items 集，再导航到首选项，并编辑 ***BorderLineStyle*** 和 ***Fill*** 属性。***Items*** 集可以通过 ***Name*** 属性以编程方式进行导航，此例中为“Germany”。

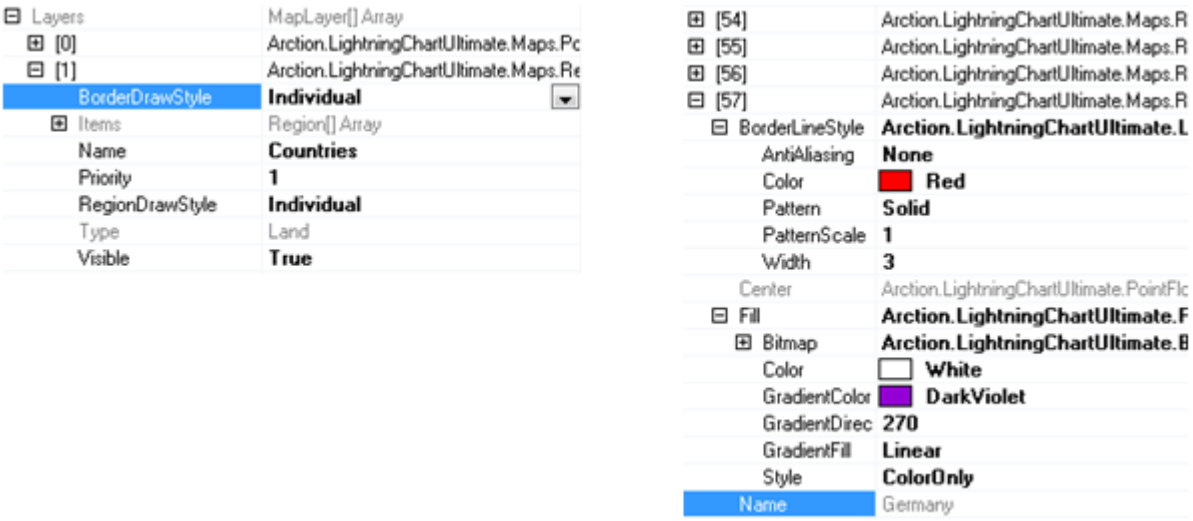


图 6-105. 设置图层边框线和区域填充样式为 Invidual，并在 Items 集中编辑区域

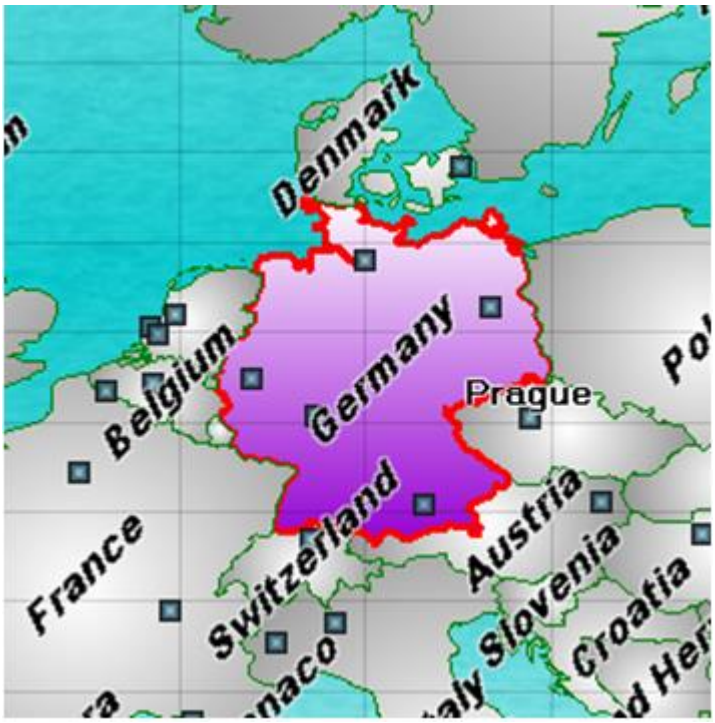


图 6-106. 采用单独填充和边界框绘制德国区域.

6.32.4 鼠标交互

开启 ***AllowUserInteraction***，以进行与地图区域和对象的各种互操作。用鼠标可以指向各种区域（陆地、湖泊）和矢量图层（河流、道路）。若设置 ***Highlight*** 为 ***Simple***，一当鼠标置于一个对象上方，用 **SimpleHighlightColor** 可将该对象变为突出显示；若设置 ***Highlight*** 为 ***Blink***，对象将会以一亮一暗颜色地闪烁。若设置 ***Highlight*** 为 ***None***，对象不会突出显示，但对象仍可以被点击，例如用来调用 ***Maps.ButtonDownOnMapItem*** 事件。

地图对象可能包含相关的数据，例如人口或其他统计数据。可用 ***UserInteractiveDeviceOverOnMapItem/UserInteractiveDeviceOverOffMapItem /ButtonDownOnMapItem***

事件处理程序来访问数据。一个地图项的数据可以使用 ***GetInfo*** 方法来检索，从而获得键和值的字典。

下面是如何在列表框中显示所有数据的一个示例。项目名称则显示在另一个文本框中。

```
    private void ButtonDownOnMap(ButtonDownOnMapItemEventArgs 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);
            }
        }
    }
}
```

6.32.5 背景图片

在 ***Maps.Backgrounds*** 属性中添加一个 ***MapBackground*** 对象，可以将位图图像显示为地图的背景。卫星图像或其他栅格图像可从几个 GIS 数据供应商处获得。图片可设置为 ***Image*** 属性，其经纬度范围可用 ***LatitudeMin***、***LatitudeMax***、 ***LongitudeMin*** 和 ***LongitudeMax*** 属性来设置。 图像不会显示在设置范围之外。

要透过地图层显示背景，需要为每一个图层调整填充设置。用透明颜色或低透明度的颜色。

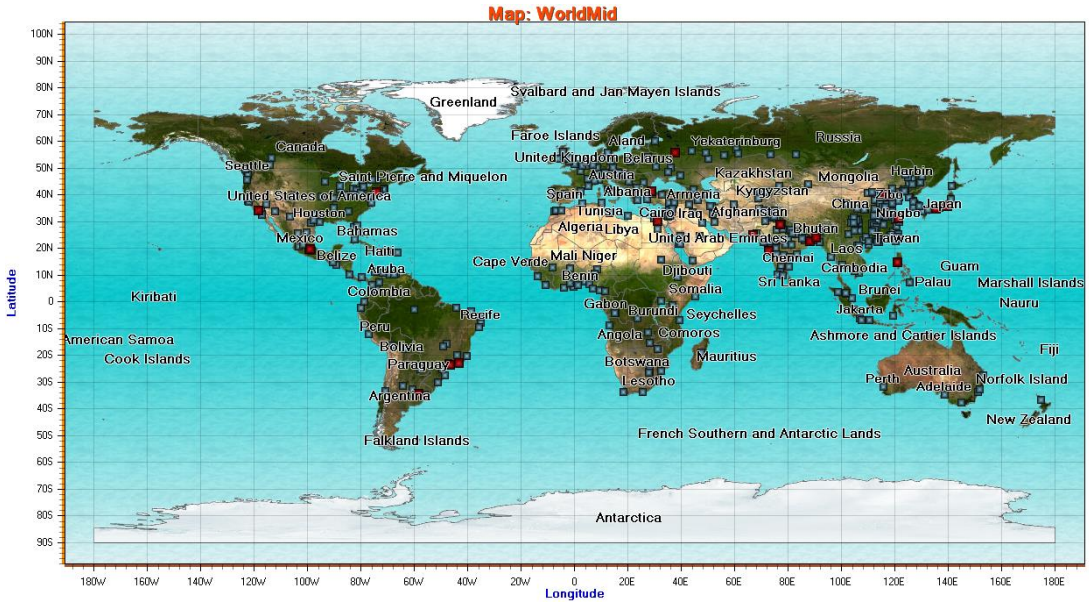


图 6-107. 世界地图。 LandOptions.FillVisible 设置为 false, 背景图片设置： 经度范围 -90...90 且 纬度范围 -180...180。地图中显示了区域边界和城市情况。

6.32.6 其他系列与地图结合

地理地图可以与任意的 ViewXY 系列类型结合。这些地图是在背景中绘制的，上面是各类系列。

图 6-108. 欧洲地图，用几个 FreeformPointLine 系列作为路线。 对其添加旗子标志作为鼠标交互的路径点。

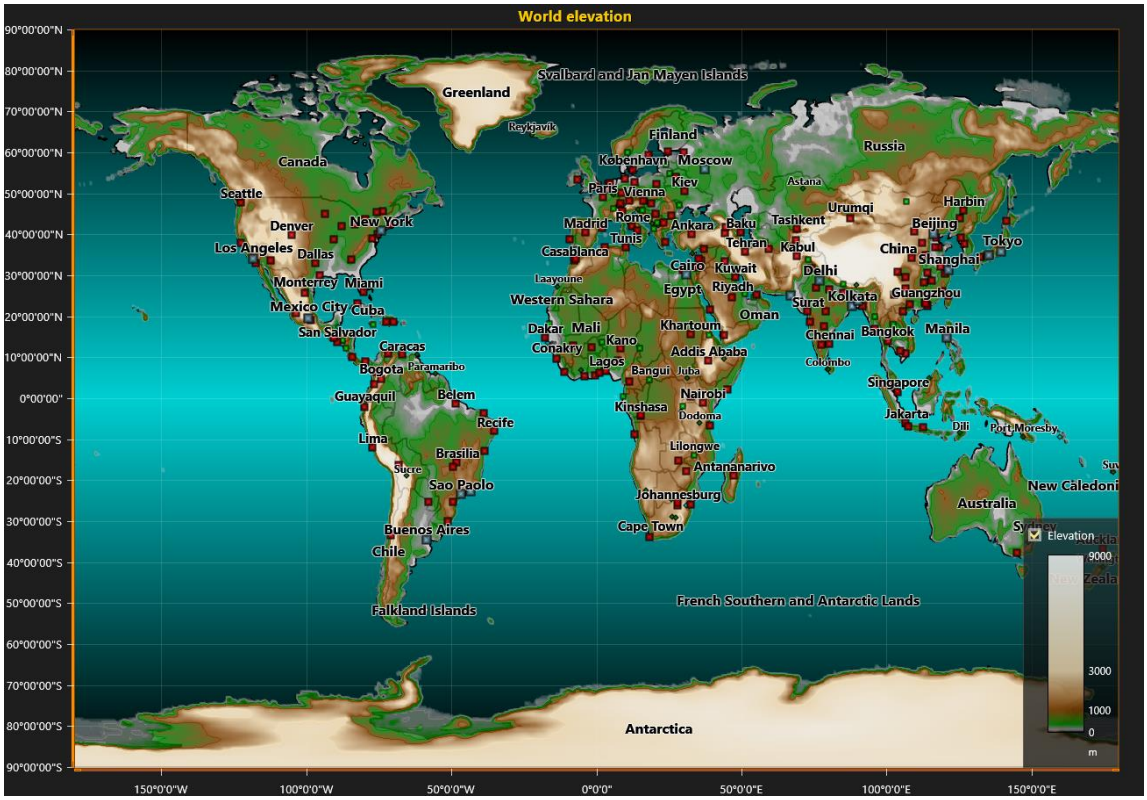


图 6-109. 世界地图，用 IntensityGrid 系列表示海拔。

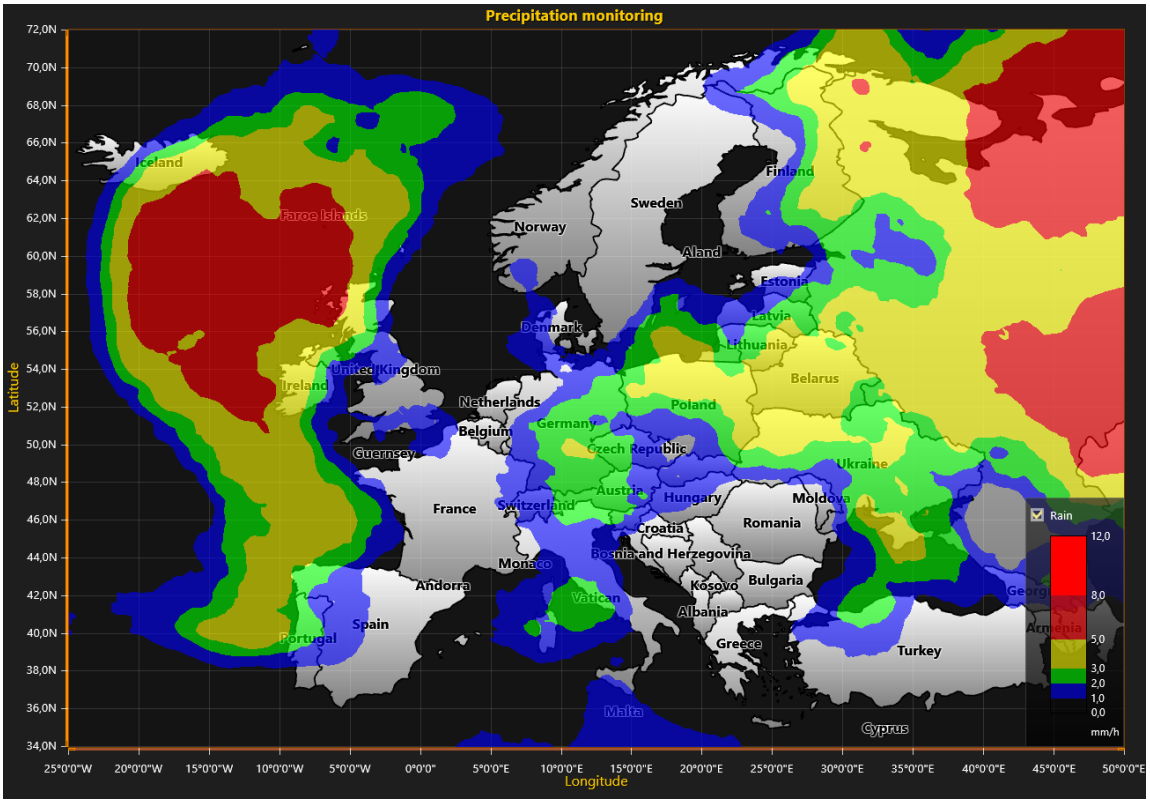


图 6-110.用 IntensityGrid 系列覆盖欧洲地图实现天气雷达数据可视化。

6.32.7 从 ESRI（美国环境系统研究所公司）图形文件数据导入地图

导入功能可根据.shp 文件来制作一幅 LightningChart 地图文件（.md）。ESRI shapefile （*.shp）是一种应用广泛的地图文件，支持矢量与多边图形数据。

用地图导向程序可将 shapefile 数据转换为 LightningChart （LC）地图格式。LC 格式支持分层，因此，多个 shapefile 文件可以合并到一个文件中。地图文件的结构与对象经过预处理可在运行时发挥最大的性能。

提示：*LightningChart® .NET 的演示应用程序有一个地图导入的示例。从其中运行导入向导程序，可通过导入来创建定制化的 LC 地图文件。*

最少三步即可完成转换：

1. 根据 Shapefile Selection Dialog 中等件来选择文件并设置图层。
2. 确定文件文本编码。
3. 选择生成的地图文件中包含的项。

注意，可为每个源 shp 文件重复第 2 和第 3 步。Shapefile 文件无法分辨其所用的编码，所以必须要用户来选择。

完成这几步后，转换开始。如果从一个定制的应用程序中导入地图，因为转换过程可能需要花费很长的时间，所以这里鼓励开发人员安装一个事件处理程序，以便用户可以获知转换过程。

此外，如果用户选择了基础层，由于预筛选数据是基于该层，那么各步骤之间可能会有相当大的延迟。

6.32.7.1 导入 shp 数据的编程界面

转换是在采用以下方法从 Maps.MapConverter 类初始化的线程上运行：

```
public bool SelectFilesAndConvert（）
```

为监控转换过程，有一个事件处理程序委托类可用：

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

对其初始化：

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

6.32.7.2 对话框

在转换过程中通常有三个对话框。另有一个不同的对话框用于选择筛选程序。

6.32.7.2.1 Shapefile Selection Dialog （Shapefile 选择对话框）

调用 SelectFilesAndConvert（ ）函数后，文件选择对话框打开。在此对话框中，用户可选择源文件并设置分层。用户也可以通过在对话框中选择适当的文件，来保存地图配置。

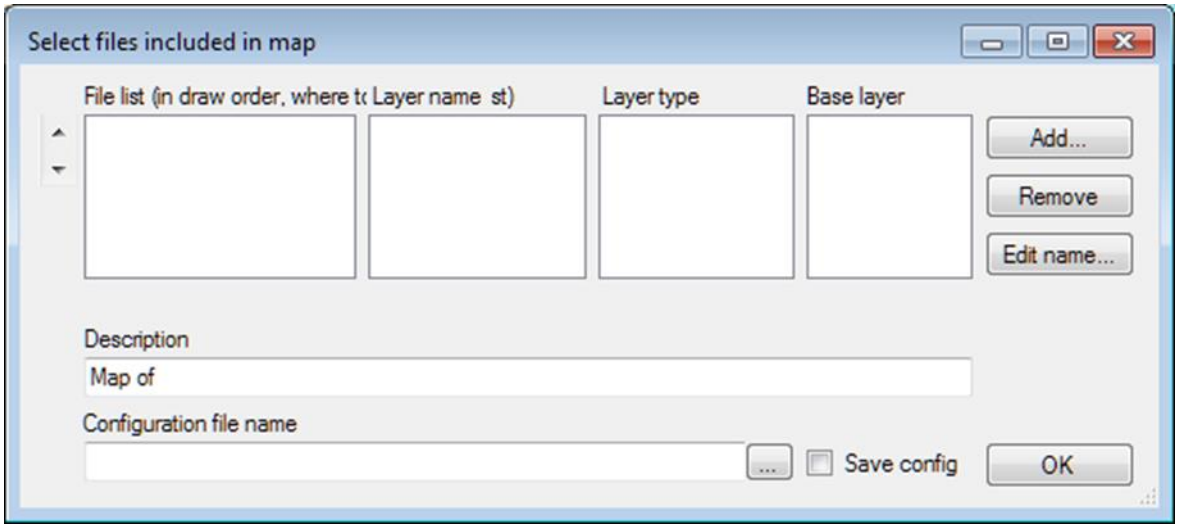


图 6-111. 源图形文件选择对话框

File list （文件列表）

包含按绘制顺序排列的文件列表。底部的文件数据最后绘制。文件的顺序可以从列表左侧的向上/向下（up/down）按钮进行更改。选择文件，然后点击向上/向下（up/down）来移动文件。

Layer name （图层名称）

图层的名称。例如“Countries”（“国家”）

Layer type 图册类型

图层类型 （规定用那种选项来渲染图层）

- City（城市）: 图层条目具有 shapefile 文件类型 POINT 性质
- Lake（湖泊）: 图层条目具有 shapefile 文件类型 POLYGON 性质
- Land（陆地）: 图层条目具有 shapefile 文件类型 POLYGON 性质
- River（河流）: 图层条目具有 shapefile 文件类型 POLYLINE 性质
- Road（道路）: 图层条目具有 shapefile 文件类型 POLYLINE 性质
- Other（其他）: 图层条目具有 shapefile 文件类型 POLYGON 或 POLYLINE 性质

Base layer （基础图层）

当用户想要一个地图，其中只包含单个/某些国家，且只有全球地图可用时，用于帅选上层图层项目选择。例如，如果图层包含多个国家，则在生成的地图中只会包含所选定的几个国家/一个国家的条目。有一个小的偏移量应用于 POINT 类型，所以如果点足够靠近边框它就会被包含进去，即使它没有和基础图层重叠。如*果所选的 shapefile 文件中的全部数据都包含于生成的文件中，不要选择基础图层，因为它会大大降低条目选择的速度，因为如果所有的条目都与基础图层重叠，就会被勾选，这是一个非常耗时的过程。*

Description （说明）
此为在地图属性中显示的自由文本。

Configuration file name （配置文件名称）
XML 格式的配置文件名称。用于导入/替换一个图层。**注意**! 当创建地图文件时使用单个文件导入。替换方法只能采用一个 shp 输入文件。

Save config （保存配置）
如果希望将地图配置保存为 xml 文件供以后使用，要勾选此选项。选择配置文件将自动设置勾选该选项。

***Add* button （*添加*按钮）**
点击以选择要添加到列表中的 shapefile 文件。

***Remove* button （*移除*按钮）**
从列表中移除所选文件。

***Edit name* button （*编辑*名称按钮）**
点击以打开“图层名称编辑器”（“Layer name editor”）。设置图层名称。

***OK* button （*确定*按钮）**
点击进入下一阶段（条目选择）。

6.32.7.2.2 Select Record Encoding and Invalid Name Fields （选择记录编码与无效名称字段）

此对话框用于选择文件文本编码和具有无效或通用名称的字段。状文件编码可能不同，而且文件中没有关于编码的信息，所以用户必须选择有效的编码。对于多项条目，条目名称可能类似于“UNK”。在此对话框中，用户可以选择清除哪个项的名称。注意，如果在下一个阶段中选择了这些条目，那么它们仍然包含在生成的文件中。

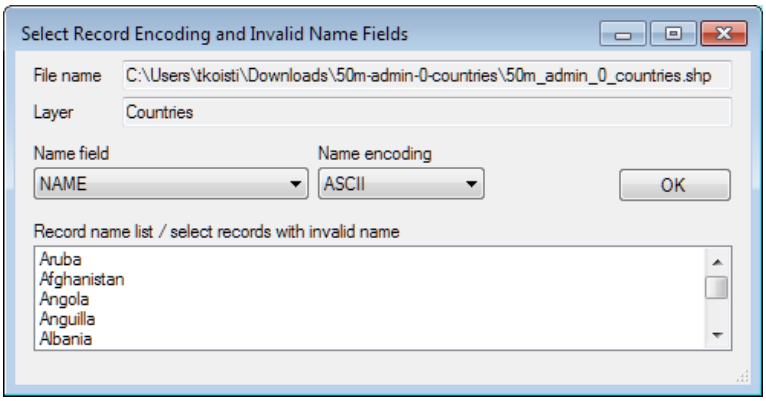


图 6-112. ‘Record encoding’ 和 ‘Invalid name’ 字段选择对话框

File name （文件名称）

编码适用的图形文件名称

Layer （图层）
图层名称。

Name field （名称字段）
图形文件中的条目名称字段。选择一个不同的字段后，列表会相应地进行更新。

Name encoding （名称编码）
条目名称编码（如果名称好像不正确，可尝试不同的值）。选择不同的编码后，列表会相应地进行更新。

Record name list / select records with invalid name （记录名称列表/选择名称无效的记录）
在“Name”字段中所选择的字段的条目列表。

OK （确定）
确认编码选择（以及可能的无效名称）。

6.32.7.2.3 Layer data selection dialog （图层数据选择对话框）

此对话框用于从图形文件中选择生产的地图文件中所包含的条目。图层名称连接到标题。对话框是自适应的，所以对于某些层来说有一些字段可以选择。例如，***River/Road*** 类型的图层，会有一个线条宽度选择，能够设置为线条宽度字段（如果适用）。注意，数据可能不包含对话框中要求的所有字段。***Name*** 字段是所有条目的必填项。

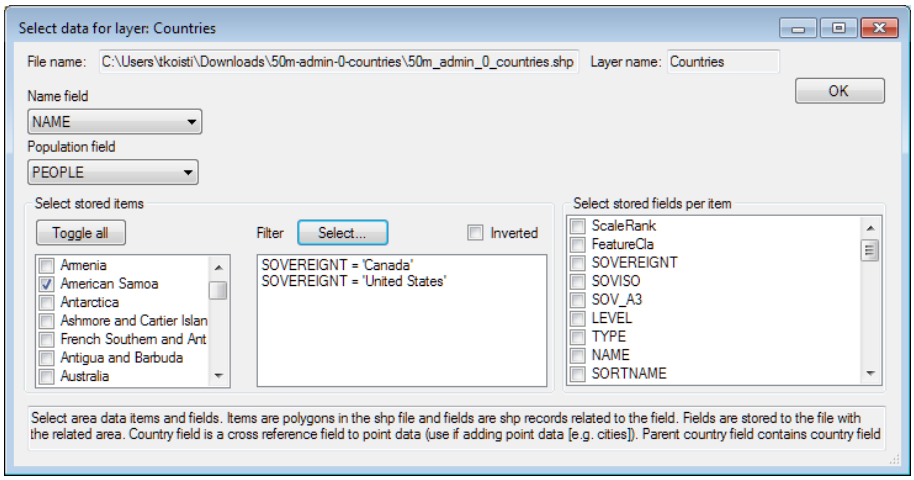


图 6-113. Layer data selection dialog（图层数据选择对话框）

对话框中可用的用户界面项:

File name （文件名称）
文件名称。

Layer name（图层名称）
图层名称。

Name field （名称字段）
用于条目名称的字段。可从编码选择对话框中自动设置，但也能在这里调整。

Population field （人口字段）
用于人口数据的字段。

Country field （国家字段）
国家名称字段。

Line stroke width field 线条笔触宽度字段
线条宽度，引导线条渲染。

Select stored items 选择存储的条目
单独选择条目，选择全部，或用 ***Filter*** 对话框来选择条目子集。

Toggle all （全选）
从文件中选择所有字段

Filter/Select... （筛选/选择…）
选择具有所选值的字段的字段。在上图中，只有 SOVEREIGNT 字段设置为“Canada”或“United States”的条目在地图中被选中。

Inverted （反选）
方向筛选选择（使用筛选程序选择的字段不包括在生产的地图文件中）。

Select stored fields per item 选择每个条目存储的字段

单击每个条目应该包含的字段。该类字段是 Dictionary 类的键值，包含每个条目的字段。

6.32.7.2.4 Item filter（条目筛选）

本对话框是从图层数据选择对话框中打开，用于为生成的地图筛选条目。

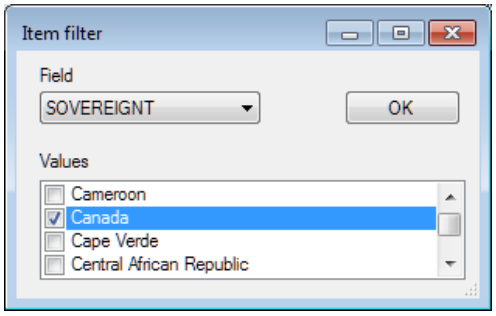


图 6-114. 条目筛选对话框

Field （字段）
选择筛选所依据的字段。

Values （值）
选择包含在生成的条目中的值。

以上选择意味着，字段名称 SOVEREIGNT 且含有值“Canada”的条目包含于生成的地图中。

6.32.8 导入与替换地图图层

用户可以导入新的图层到地图中，并替换现有的图层。有四个方法可以从 Maps 界面中导入并替换一个图层到地图中。这在软件的应用程序运行时，检索频繁更新的 shp 数据时非常有用。

ImportNewLayer 方法可将一个新的地图图层插入到给定的层索引中，***ImportReplaceLayer*** 方法可替换在给定图层索引处的地图图层。

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

此处 ***shpFilename*** 是源 shp 文件名的名称，***targetLayerIndex*** 是新图层的索引。此方法使用上面所述对话框来设置地图配置。

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

此处 ***shpFilename*** 是源 shp 文件名的名称，***targetLayerIndex*** 是新图层的索引，***configFile*** 是地图配置文件名称。此方法使用上面所述的对话框创建的配置文件。

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

此处 ***shpFilename*** 是源 shp 文件名的名称，***targetLayerIndex*** 是新图层的索引。此方法使用上面所述对话框来设置地图配置。

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

此处 ***shpFilename*** 是源 shp 文件名的名称，***targetLayerIndex*** 是新图层的索引，***configFile*** 是地图配置文件名称。此方法使用上面所述的对话框创建的配置文件。

配置文件是一个纯 xml 文件，可以用文本编辑器编辑，但不建议编辑。

6.33 Tile maps （瓦片地图）

演示示例： HERE Maps streets; HERE Maps satellite; HERE Maps with small charts

LightningChart 支持以下在线瓦片数据服务：

- Here: 街道地图，卫星图像

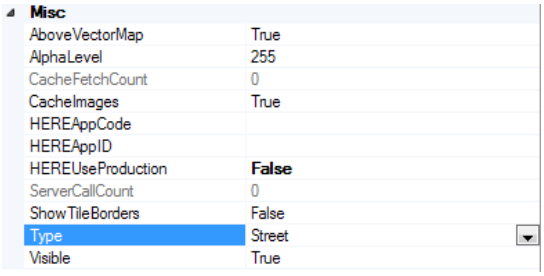


图 6-115. TileLayer 图层的属性

在 **ViewXY.Maps.TileLayers** 集中添加 ***TileLayer*** 对象。用 ***AlphaLevel*** 属性可以插入多个图层并设置为半透明。***TileLayer*** 对象依照在 ***TileLayers*** 集中出现的顺序进行渲染，第一个图层位于背景中。设置 ***AboveVectorMap*** = ***False*** 后，如果定义了矢量地图，则该图层在矢量地图之前渲染（参阅第 **Error! Reference source not found.**章节）。默认情况下， ***TileLayer*** 在矢量地图之后渲染。

TileLayer 以小图像形式通过 http 协议从在线服务供应商获取信息，并在图表区域显示它们。这些图像将在缩放或平移地图视图时刷新。

加载一组新的 **Tile** 图层需要花些时间，最多要几秒钟。

Tile cache（Tile 缓存）

图表将 Tile 存储于一个缓存文件中，当在同一区域进行频繁平移或缩放时可大大降低加载时间。当图表需要显示某个 Tile 时，首先检查是否可以在缓存文件夹中找到它，如果不能，则从 web 服务中检索。在团队使用中，许多工作站需要访问瓦片地图，明智的做法是选择一个共享的本地网络服务器文件夹，缓存文件夹是 ***c:\Users\[Current user]\AppData\Local\Temp***。

在 ***ViewXY.Maps.TileCacheFolder*** 中设置缓存文件夹。

调用 ***ViewXY.Maps.ClearTileCacheFolder（）***方法来清除缓存文件夹。

6.33.1 HERE

LightningChart 通过 Here 来支持瓦片数据服务。开发商或者终端用户必须与 Here 签订独立的合同，才能使用 Here 服务器。免费试用密钥可从 https://developer.here.com/plans/api/consumer-mapping 获取。

Selecting type （选择类型）

设置 ***TileLayer.Type = Street*** 来使用街道地图。街道地图可以放大到非常近。

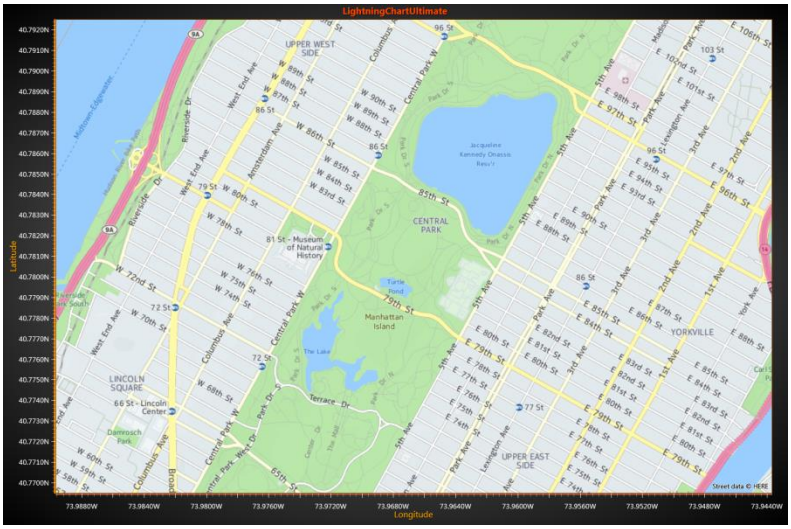


图 6-116. TileLayer.Type = Street.

设置 ***TileLayer.Type = Satellite*** 来使用卫星图像。

图 6-117. TileLayer.Type = Satellite.

显示系列和其他图表元素，比如注释（如适用）。

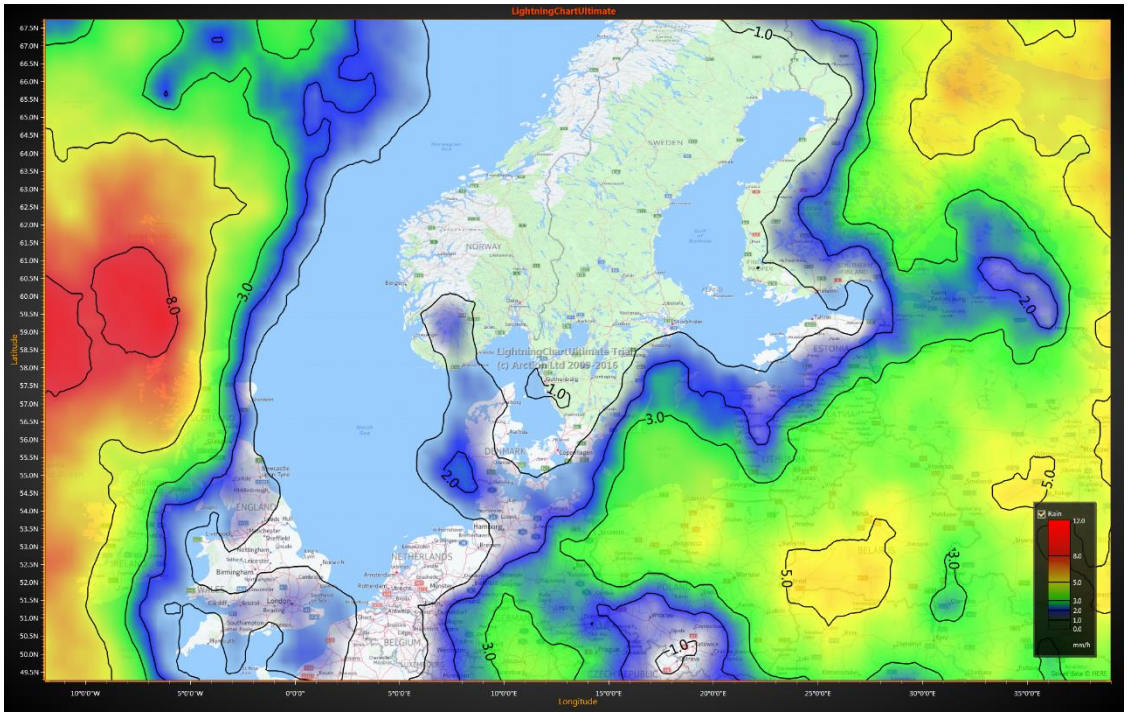


图 6-118. 街道地图用 IntensityGridSeries 显示天气数据。

6.34 StencilAreas

演示示例： Maps with intensity series stencil; Chromaticity diagram, Silicon wafer map analysis （仅 WinForms）

IntensityGridSeries、***IntensityMeshSeries*** 和 ***Maps*** 具有 ***StencilArea*** 功能，可在绘制的数据区域内或外进行屏蔽。例如，如果数据显示在地图上，模板可以用来将可见数据限制在例如国家等特定的地图区域。通过创建一个新的 ***StencilArea*** 对象可以应用 ***StencilArea***，然后通过 ***AddPolygon*** （）定义其规格为 PointDouble2D 数组，或通过 ***AddMapLayerIndex*** （）定义为一个地图图层，最后将它们添加到应该屏蔽的系列中。

StencilAreas 有两种类型：

- ***AdditiveAreas*** 创建一种阳掩模板——仅绘制区域内部的数据，外部的则裁剪掉。
- ***SubtractiveAreas*** 创建一种阴掩模板——区域内部的数据会被裁剪掉，而绘制外部的数据。注意，***SubtractiveAreas*** 只可与 ***AdditiveAreas*** 一起使用，没有二者不能进行裁剪。

每当向列表（***AdditiveAreas*** 或 ***SubtractiveAreas***）中添加一个 ***StencilArea*** 对象，应该为各系列分别调用 ***InvalidateStencil*** （）或 ***InvalidateData*** （）。另外还建议将定义模板的点的数组按顺时针顺序设置。

6.34.1 AdditiveAreas

用 ***AdditiveAreas*** 来定义需要绘制的区域。外部的都会被剪裁掉。

```
// 为IntensityGrid定义一个累加的 StencilArea

PointDouble2D[] stencilPoints = new PointDouble2D[] {
        new PointDouble2D（30, 5）,
        new PointDouble2D（30, 95）,
        new PointDouble2D（195, 95）,
        new PointDouble2D（195, 5）
};
StencilArea stencilArea = new StencilArea（_intensityGrid.Stencil）;
stencilArea.AddPolygon（StencilPoints）;
_intensityGrid.Stencil.AdditiveAreas.Add（stencilArea）;
_intensityGrid.InvalidateStencil（）;
```

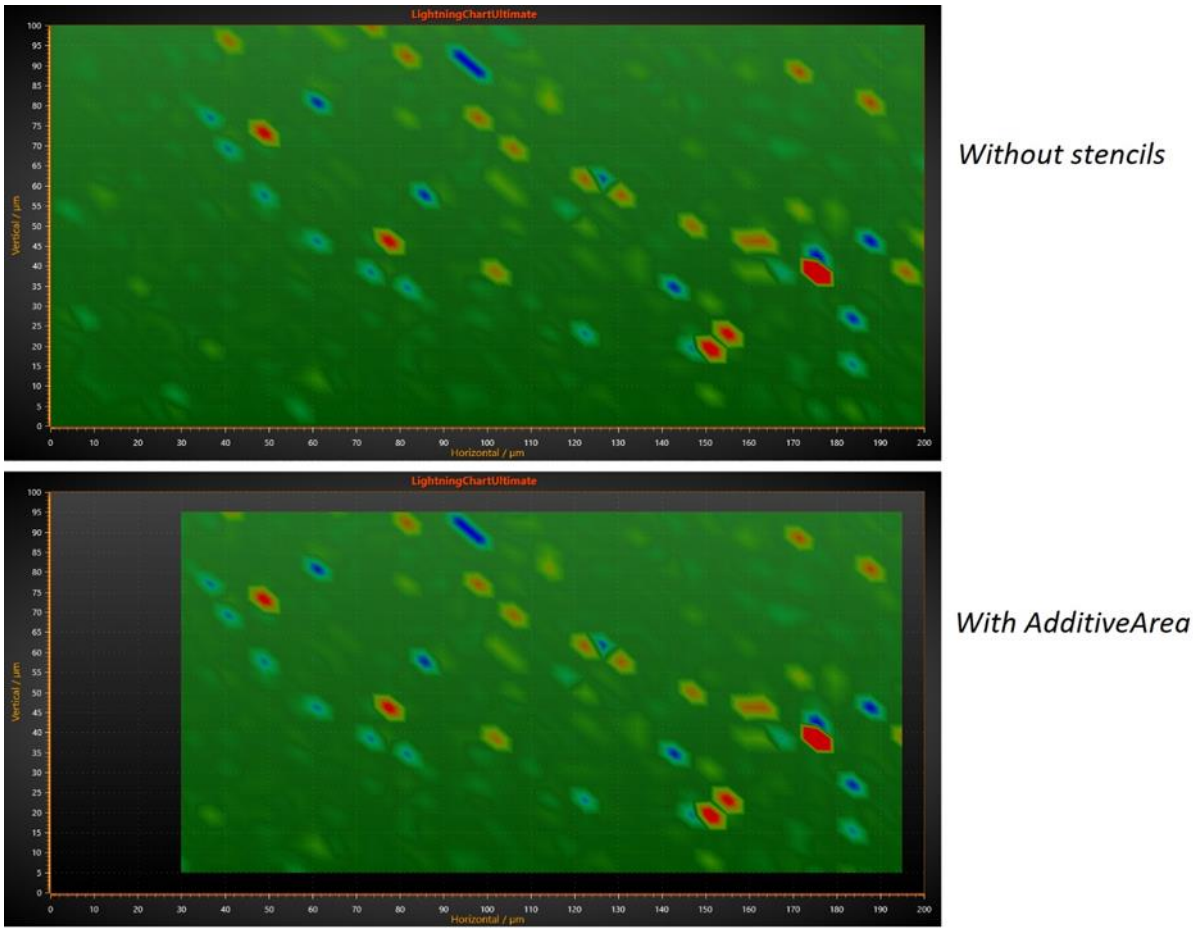


图 6-119. 顶部位不带有任何模板的 IntensityGrid；底部，同一个网格带有使用上述代码创建的 AdditiveArea。

6.34.2 SubstractiveAreas

用 ***SubstractiveAreas*** 来定义应绘制于 ***AdditiveArea*** 内部的区域

```
// 对IntensityGrid定义两个差减的StencilAreas

PointDouble2D[] pnt2 = new PointDouble2D[] {
        new PointDouble2D（130, 70）,
        new PointDouble2D（130, 90）,
        new PointDouble2D（160, 90）,
        new PointDouble2D（160, 70）,
};
```

```
StencilArea stencilArea2 = new StencilArea（_heatMap.Stencil）;
stencilArea2.AddPolygon（pnt2）;
_heatMap.Stencil.SubtractiveAreas.Add（stencilArea2）;
_heatMap.InvalidateStencil（）;

PointDouble2D[] pnt3 = new PointDouble2D[] {
        new PointDouble2D（50, 10）,
        new PointDouble2D（50, 25）,
        new PointDouble2D（90, 25）,
        new PointDouble2D（90, 10）,
};
StencilArea stencilArea3 = new StencilArea（_heatMap.Stencil）;
stencilArea3.AddPolygon（pnt3）;
_heatMap.Stencil.SubtractiveAreas.Add（stencilArea3）;
_heatMap.InvalidateStencil（）;
```

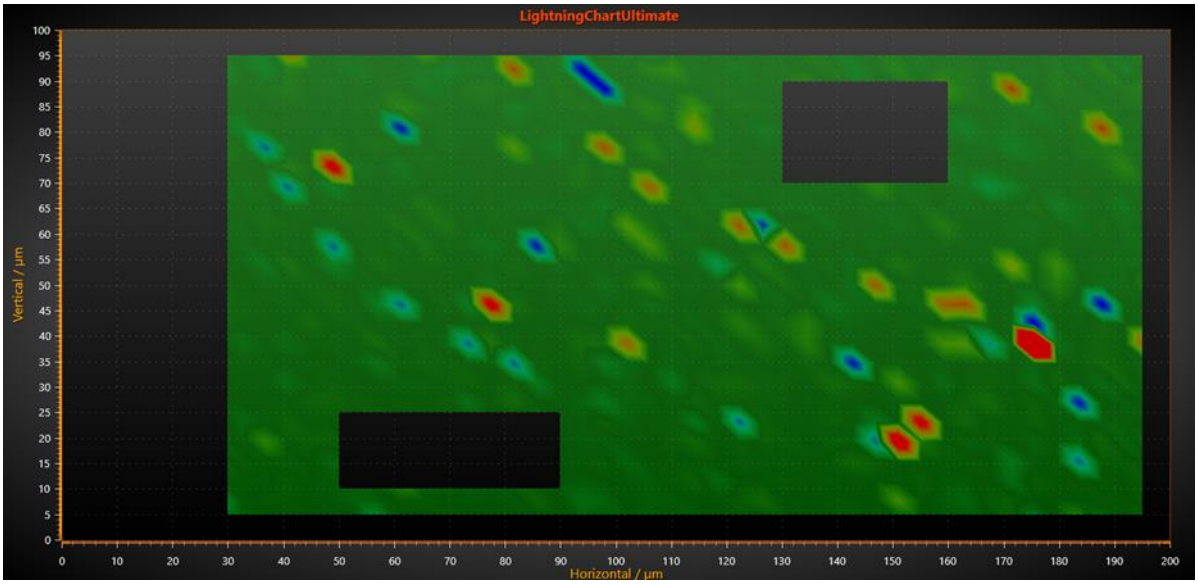


图 6-120. 用上述代码设置带有两个 SubstractiveAreas 的 IntensityGrid。 注意，在使用 SubstractiveAreas 之前必须先设置一个 AdditiveArea

6.34.3 多重 StencilAreas

可以设置多重 ***StencilAreas***，累加和差减两种都有。如果有两个或两个以上的区域重叠，则将这些区域连接起来。

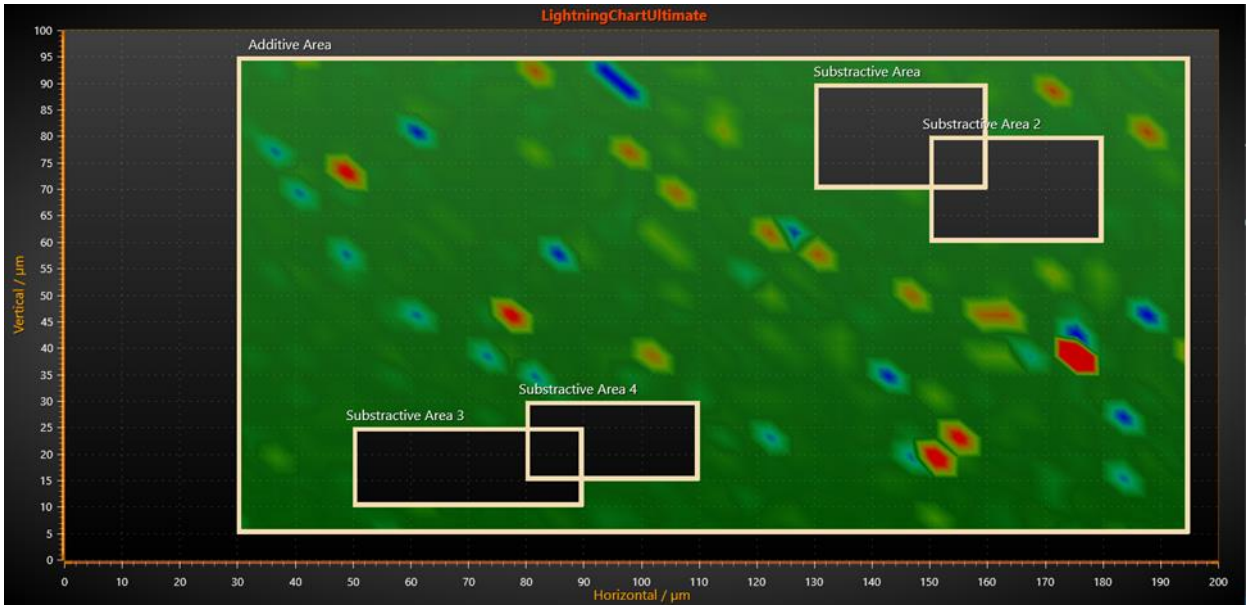


图 6-121. 采用多重 StencilAreas；一些 SubstractiveAreas 重叠起来，这样区域便连接在一起。另外还绘制有带有可见边框的透明多边形来标记模板的位置。

6.35 Data cursors

从 10.4 版本开始，LightningChart 有一个内置的数据游标，它会自动跟踪最接近鼠标光标的系列值，并将其显示在结果表中。 光标由水平和垂直头发交叉线，跟踪点位于最接近数据值的位置，轴标签显示当前的 X 和 Y 值，以及结果表，除了轴值外，还显示了系列名称及其颜色。

可以通过 Visible 属性启用或禁用数据游标。 也可以隐藏一些光标通过设置 ShowHaircrossLines 或 ShowLabels 单独设置线条或轴标签等组件，或其他基于应隐藏内容的相应“显示”属性，false。 的外观 cursor 也可以通过组件特定的属性进行修改。 LabelFont 修改轴标签文本，LineStyle 可用于自定义头发交叉线，TrackingPointStyle 允许更改跟踪点。 Results 属性包含修改结果表的所有选项。

```
// Enables data cursor but hides its axis labels.
_chart.ViewXY.DataCursor.Visible = true;
_chart.ViewXY.DataCursor.ShowLabels = false;
// Modifying result table.
_chart.ViewXY.DataCursor.Results.Background.Color = Colors.DarkBlue;
```

DataCursor	
Configuration	
XAxisLabelBackground	Black
XAxisLabelBorderColor	White
XAxisLabelForeground	Yellow
XAxisLabelUseSeriesColor	True
XAxisLabelVisible	True
XAxisLineVisible	True
YAxisLabelBackground	Black
YAxisLabelBorderColor	White
YAxisLabelForeground	Yellow
YAxisLabelUseSeriesColor	True
YAxisLabelVisible	True
YAxisLineVisible	True
LabelFont	Segoe UI, 12pt
LineStyle	
RealTimeTracking	False
Results	
ShowColorIndicator	True
ShowHaircrossLines	True
ShowLabels	True
ShowResultTable	True
ShowTag	False
ShowTrackingPoint	True
SnapToNearestDataPoint	False
strTag	Tag
TrackingPointStyle	
Visible	True

Figure 6-122. 数据游标的属性树。

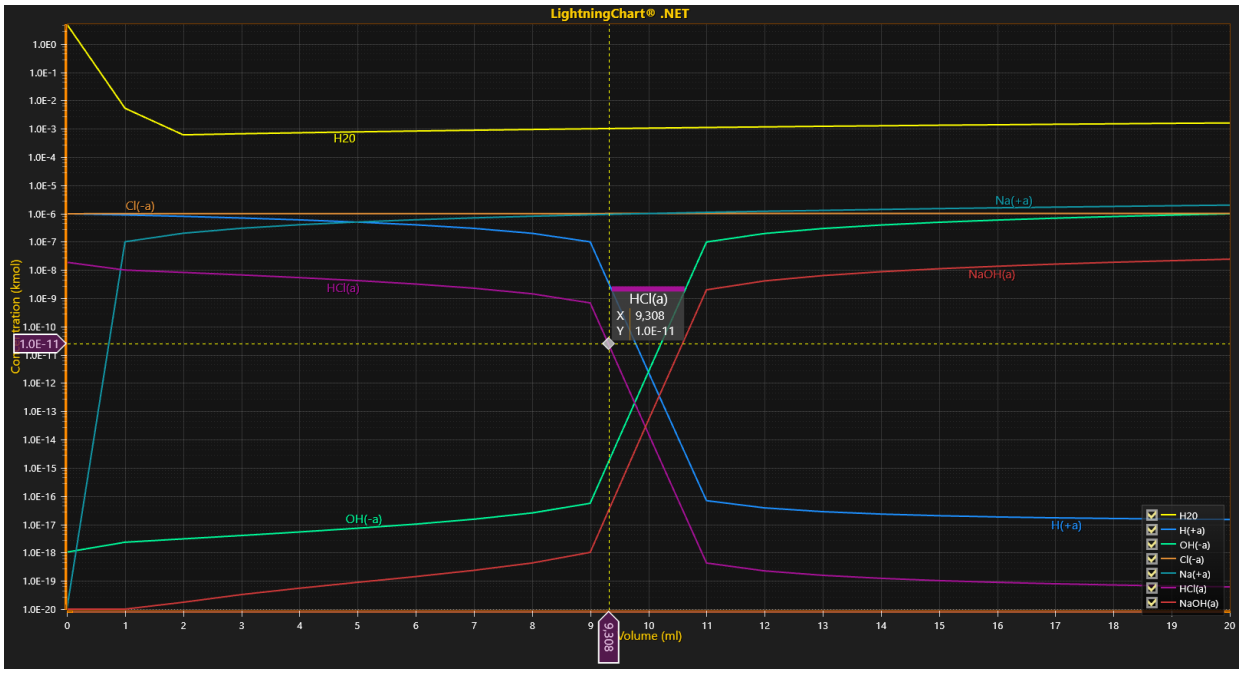


Figure 6-123. 数据游标已启用。 没有隐藏任何单独的组件.

数据游标会根据跟踪的系列是否具有可见线或仅具有可见数据点（散点图）来更改其行为。 如果该线可见，光标会在光标当前的 X 位置垂直找到最近的系列及其值。 如果没有可见的线，则光标会在任何方向上追踪最近的数据点。 启用 SnapToNearestDataPoint 会覆盖此设置，使光标始终在任何方向上找到最近的实际数据点值。

应该注意的是，在要求严格的实时应用程序中使用数据游标会显着降低性能，因为游标不断地计算最近的数据值。 为了解决这个问题，游标具有 RealTimeTracking 属性，启用后可以提高整体性能。缺点是光标的更新频率较低，并且可能看起来很滞后，尤其是当鼠标在滚动或扫描图表上移动时。

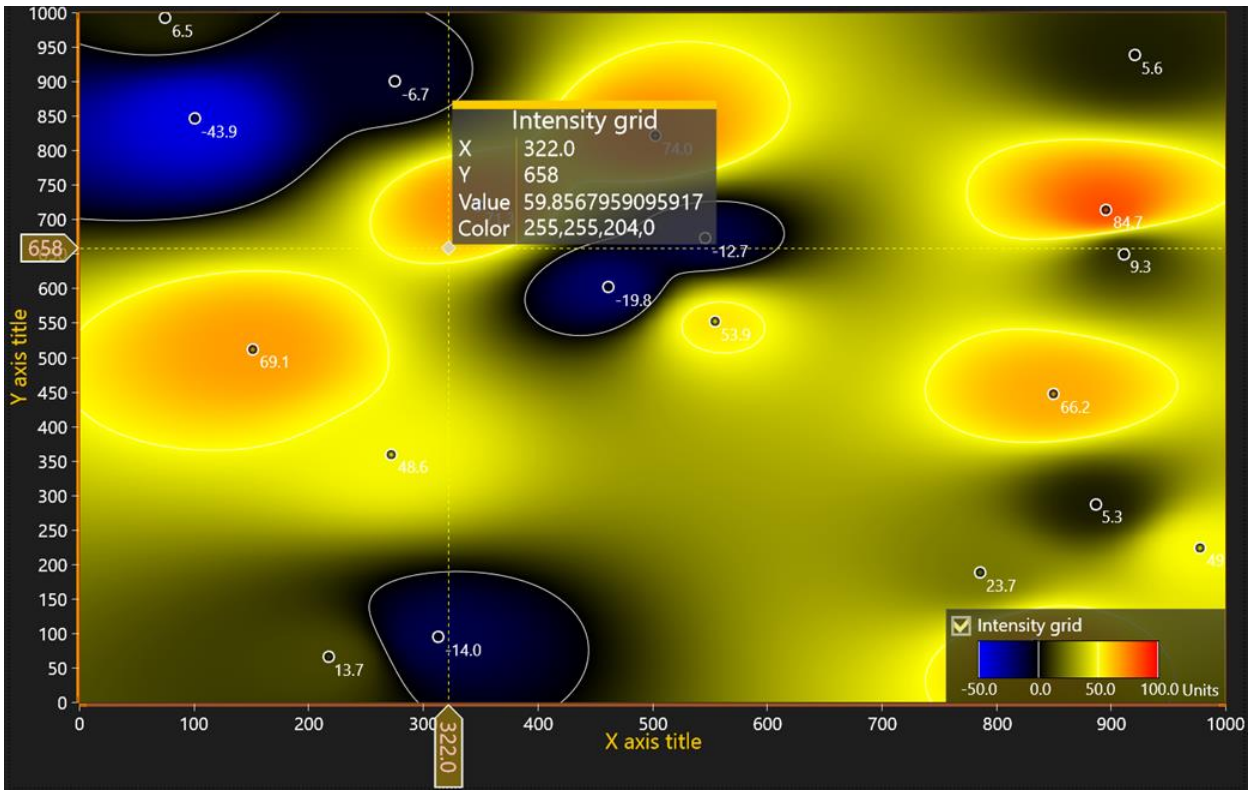


Figure 6-124. 数据光标跟踪强度网格。 除了 X 和 Y 值外，光标还显示网格的值和颜色字段

6.36 LineSeriesCursors

演示示例： Point line; Multi-channel cursor tracking; Segments with splitters; Logarithmic axes

线条系列游标可通过追踪 X 坐标值来对线条系列数据进行可视化分析。系列值只能用实现 ***ITrackable*** 接口（***SampleDataSeries***、***SampleDataBlockSeries***、 ***PointLineSeries***、***AreaSeries***、***HighLowSeries***）的系列来求解。对于其他系列类型而言，光标不会自动追踪 Y 轴坐标。

将 ***LineSeriesCursor*** 对象添加到 ***LineSeriesCursors*** 集中。开启 ***SnapToPoints*** 可将光标从一点跳到另一点。用 ***Style*** 属性可设置光标追踪类型。当设置 ***Style*** 为 ***PointTracking*** 后，可以使用任何追踪

点的类型，甚至是位图图像。若使用 ***HairCrossTracking*** 类型，在线系列点的 Y 值处会绘制一条水平线。如果同一系列的多个点同时碰到光标位置，则线条绘制在最小点和最大值点的中间。

图 6-121. 线条系列游标：垂直点追踪光标（青色的线和十字），十字叉线追踪光标（红线）以及不带追踪的垂直全长光标（黄线）

开启 ***IndicateTrackingYRange*** 后，会绘制一条水平柱状，从碰到光标中间的点的最小值延伸到最大值。

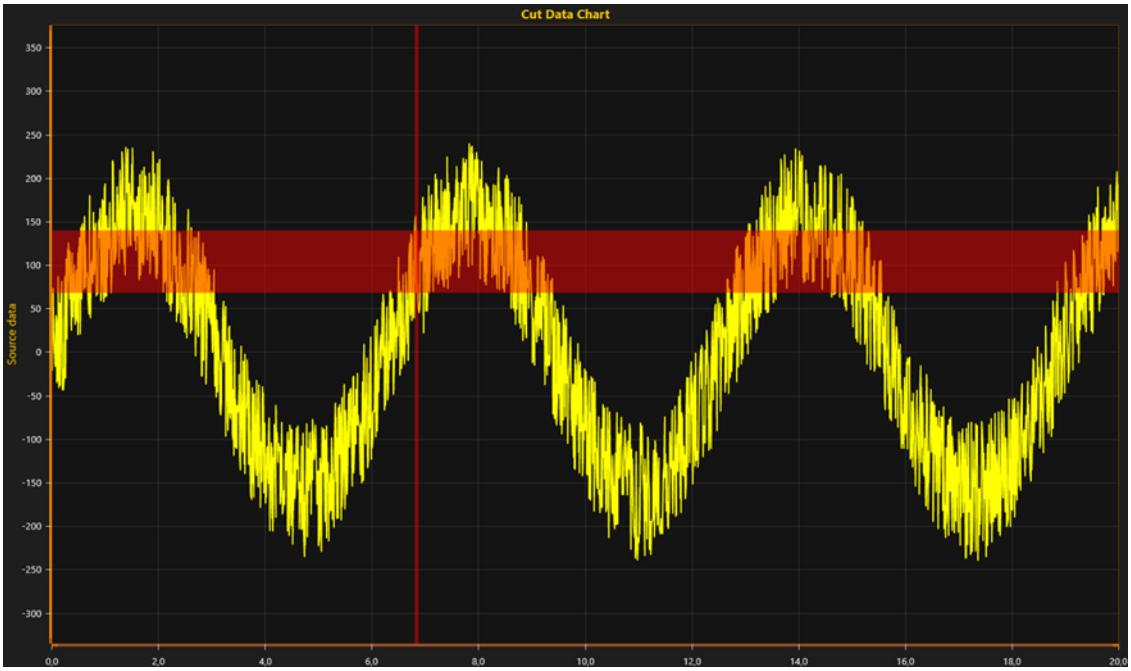


图 6-122. 用 Y 轴区域指示的十字叉线光标（IndicateTrackingYRange = true）.

6.36.1 解析 LineSeriesCursor 位置处的数据值

演示示例： Multi-channel cursor tracking

通过 X 屏幕坐标或 Y 轴值可求解实现 ***ITrackable*** 接口的系列。The series implementing ***ITrackable*** interface can be solved by X screen coordinate or by X axis value.

可跟踪的系列有精确和粗略两种数值求解方法。如果需要，精确的方法 ***SolveYValueAtXValue*** 循环遍历数据点并找到最近的数据点匹配。粗略法 ***SolveYCoordAtXCoord*** 使用缓存的系列渲染数据来求解匹配的 Y 屏幕坐标。

6.36.1.1 精确方法，用数据点数组根据 X 值求解 Y 值

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

if （result.SolveStatus == LineSeriesSolveStatus.OK）
{
        //PointLineSeries在同一个X值可能有两个或多个点，如果这样，将其置于最小与最大中心
      yValue =（result.YMax + result.YMin）/ 2.0;
      return true;
}
```

注意! 当禁用 ***cursor.SnapToPoints*** 后， ***SolveYValueAtXValue*** 返回与其相邻的点之间的内插值（光标线和系列线的交叉点）

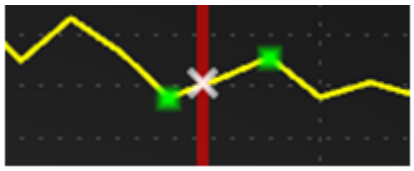


图 6-123. 禁用 SnapToPoints 后，SolveYValueAtXValue 内插入相邻数据点之间的值

6.36.1.2 粗略方法，用数据点数组根据 X 坐标求解 Y 屏幕坐标

```
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;
        }
}
```

当系列包含很多数据点时，假设 > 100.000, 那么用粗略方法通常要快得多。如果图表的大小或 Y 段的像素高度很低，这种粗略方法可能非常不准确。

通过调用 X 和 Y 轴的 ***CoordToValue*** 方法，粗略方法的屏幕坐标可转换为轴值。

使用一个 AnnotationXY 对象显示在光标旁边的值是一个不错的方法，可参阅第 6.26 章节。

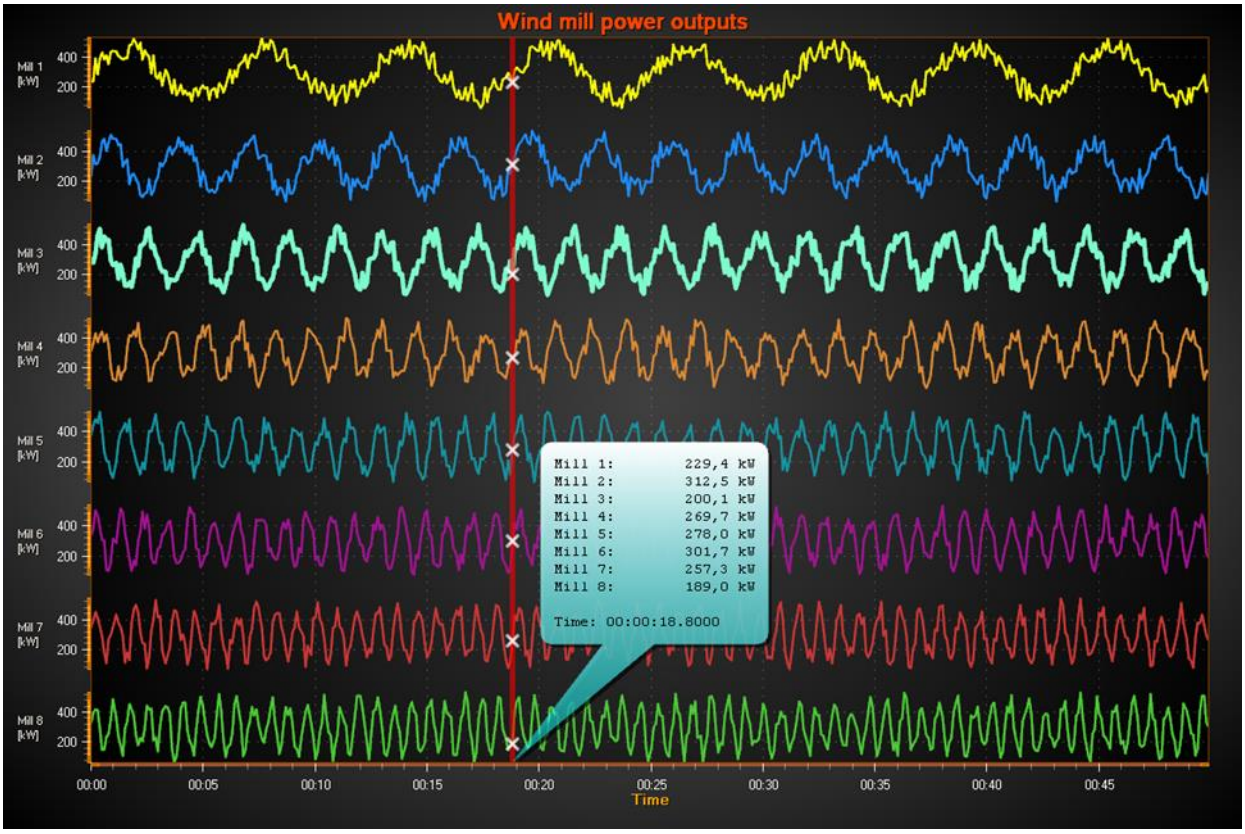


图 6-124. LineSeriesCursor 用于追踪 PointLineSeries；值显示在一个 AnnotationXY 对象中。

6.36.2 从 FreeformPointLineSeries 中求解数据值

FreeFormPointLineSeries 实现精确和粗略的数值求解方法，非常类似于用以求解在 ***LineSeriesCursor***（*用于实现* ***ITrackable*** *接口的系列*）位置的数据值的方法。

注意! FreeformPointLineSeries 无法执行 ***ITrackable*** 界面，因此无法用 ***LineSeriesCursor*** 追踪。

对于给定的 X 轴值，可用精确方法 ***SolveYValuesAtXValue*** 求解所有的 Y 值，并返回一个 ***LineSeriesValueSolveResult*** 结构的可迭代列表。

```
IList<LineSeriesValueSolveResult> results = series.SolveYValuesAtXValue(value);
foreach (LineSeriesValueSolveResult result in results)
{
    if (result.SolveStatus == LineSeriesSolveStatus.OK)
    {
        // Do.
    }
    else if (result.SolveStatus == LineSeriesSolveStatus.NoPointsFound)
    {
        // Do.
    }
}
```

对于给定的 X 屏幕坐标，可用粗略方法 ***SolveYCoordsAtXCoord*** 求解所有 Y 屏幕坐标，并返回一个 ***LineSeriesCoordinateSolveResult***-结构的可迭代列表。

```
IList<LineSeriesCoordinateSolveResult> results = series.SolveYCoordsAtXCoord(fCoordX);
foreach(LineSeriesCoordinateSolveResult result in results)
{
    if(result.SolveStatus == LineSeriesSolveStatus.OK)
    {
        // Do.
    }
    else if(result.SolveStatus == LineSeriesSolveStatus.NoPointsFound)
    {
        // Do.
    }
}
```

如果给定的 X 轴值或 X 屏幕坐标在点之间碰上，结果将由插值值/坐标构成。

6.37 EventMarkers

演示示例： Tracking markers; Map route; Heatmap color spread; Segments with splitters; Bubble chart; Campbell diagram; Curve node editing

EventMarkers 可以标记一个感兴趣的点，这里在实时监控期间发生了一些特殊的事情，或者只是想用一种特殊的注释来标记一段数据。 用 ***Symbol*** 属性可定义标记符号，用 ***Label*** 属性可定义文本标签。用属性可设置垂直位置，如果需要，可使用 ***Offset*** 移动对象属性。所有标记需用 ***XValue*** 来分配，以设置标记在 X 轴上的位置。

设置 ***Symbol.Shape*** 可选择标记的形状。可用的形状包括有：***Rectangle***、***Circle***、***Triangle***、***Flag***、 ***FlagLightning***、***Cross***、***CrossAim***、***Bitmap***、***HollowBasic***、***HollowBasicActive***、***HollowHarmonic***、***HollowActiveSideband***、***HollowSideband***、***HollowTailedActive*** 和 ***HollowTailed***。

图 6-125. 在贸易示例中的标记（Markers）

6.37.1 图表事件标记

ChartEventMarker 集可添加图表标记。用图表标记可指示一个兴趣点，例如 “Test person stood up”、“Capacitor bypassed”。图表事件标记与系列事件标记不同，并不依附于一个特定的系列。 这种标记可用鼠标拖动到另一个位置里。

通过 ***VerticalPosition***、***XValue*** 和 ***Offset*** 属性，可设置图表标记的位置。此外，设置 ***BindToXAxis*** 为 true 可将标记绑定到一特定的 X 轴。在实践中，这会使标记停留在当前的 x 值上，并在例如 x 轴平移时移动。当 ***BindToXAxis*** 禁用后，不管轴如何移动，标记将留在同样的图表位置。如果有数条 X 轴，用 ***AssignXAxisIndex*** 可以设置标记绑定到哪条轴上。

ClipInsideXRange 可与禁用的 ***BindToXAxis*** 一起使用。当标记所绑定的特定 X 轴值不在可见的 X 轴值域内时，这种用法可以创建一个用以剪切的标记，该 X 值可通过 ***XValue*** 属性来设置。用鼠标手动移动标记将禁用此设置。还有 ***ClipInsideGraph*** 属性，可决定是否能在图形区域之外绘制图表标记。

6.37.2 线条系列事件标记

线条系列具有 ***SeriesEventMarkers*** 集属性，可用于分配系列特定的事件标记。该系列事件标记可用鼠标拖动到另一个位置，同时将该事件标记依附到系列值。当禁用此属性后，标记的 ***VerticalPosition*** 必须设置为 ***TrackSeries***。这可用于系列来实现 ***ITrackable*** 接口。

设置 ***HorizontalPosition*** 为 ***SnapToPoints***，标记将自己水平对齐到最近数据点的位置。若设置 ***HorizontalPosition*** = ***AtXValue*** ，可将标记置于任意的 X 值；另外，设置 ***VerticalPosition*** = ***AtYValue***，可以设置标记垂直对齐到任意的 Y 高度上。

除了正常的形状组， ***SeriesEventMarker*** 还支持两种特别的 **Symbol.Shape** 设置，***HollowYAxis*** and ***HollowYAxisActive,*** 可以生成带有 Y 轴刻度投射的垂直线。他们具有一个像素宽的垂直线，该线是从系列所依附的 Y 轴上选择 ***MajorTicks*** 和 ***MinorTicks*** 的位置。若要调整刻度长度，可编辑 ***YAxis.MajorDivTickStyle.LineLength*** 和 ***YAxis.MinorDivTickStyle.LineLength*** 属性。

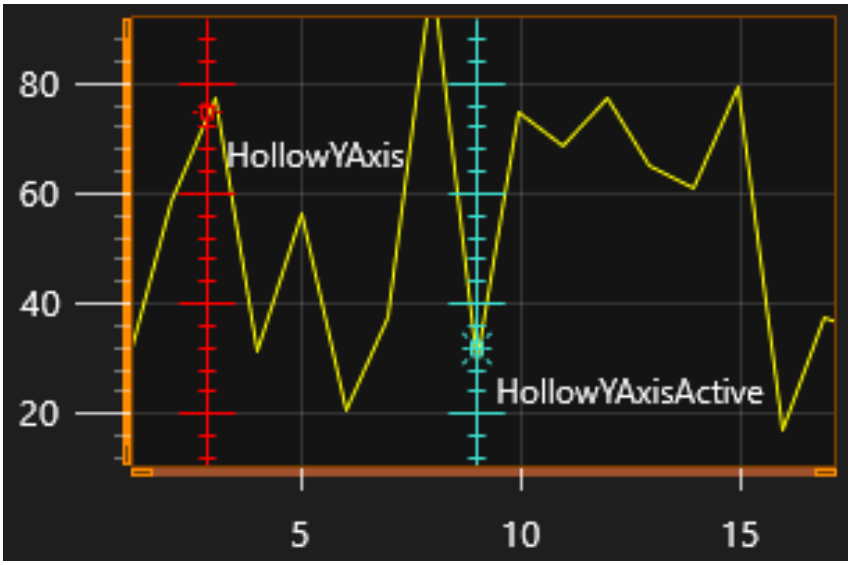


图 6-126. 两个特别的 SeriesEventMarkers 形状： HollowYAxis 和 HollowYAxisActive； 在制作每个系列的数据游标时非常方便。

6.38 持续的系列渲染图层

演示示例： Lines / points; Areas /high-lows

PersistentSeriesRenderingLayer 可用于非常快速渲染重复的线/点数据，或在相同的 X 和 Y 值域内反复绘制的线/点/高低/区域填充数据。

例如，看下 FFT 监控的情况：每秒钟收到 20 个新的数据条。最新的数据应该与所有的历史轨迹一样可见。但是监控会持续数小时。通过用常规渲染法来渲染这种数据，每小时则需要 20 * 60 * 60 = 72000 个新线条系列。计算机可能会在还没监控 1 小时便耗尽内存。可以肯定的是渲染会变得非常慢，以至于再不能起作用。

PersistentSeriesRenderingLayer 是一种位图，可循序渐进地添加渲染数据。如果没有清除指令，可以一直存储图形。这样，每一轮渲染，图层上只会渲染一个系列，随后进行屏幕上的图层渲染。CPU 负载或内存占用不会增加。如果现有的数据应该逐渐消失，可以通过增加位图像素的透明度来实现。

根据需要可以创建任意数量的 ***PersistentSeriesRenderingLayer*** 对象，在每一轮更新中，任何的系列量都可以在它们每个上面渲染。

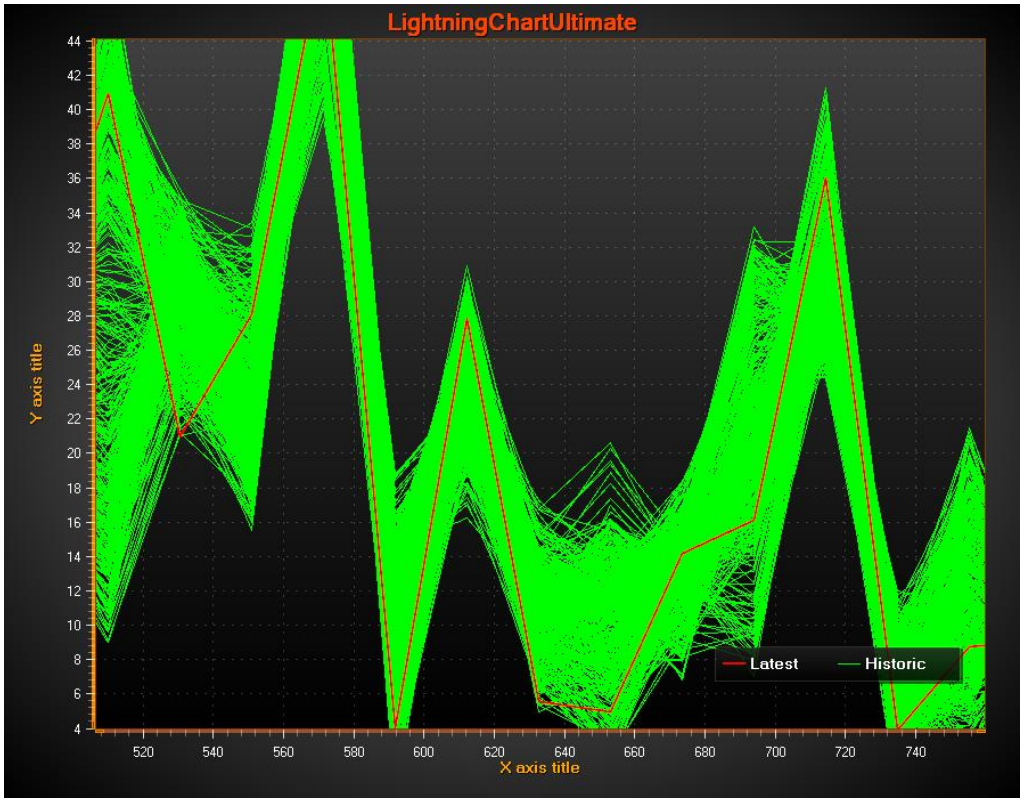


图 6-127. 持续性图层可显示历史轨迹（绿色）。其上方显示的是一个普通的 PointLineSeries（红色）

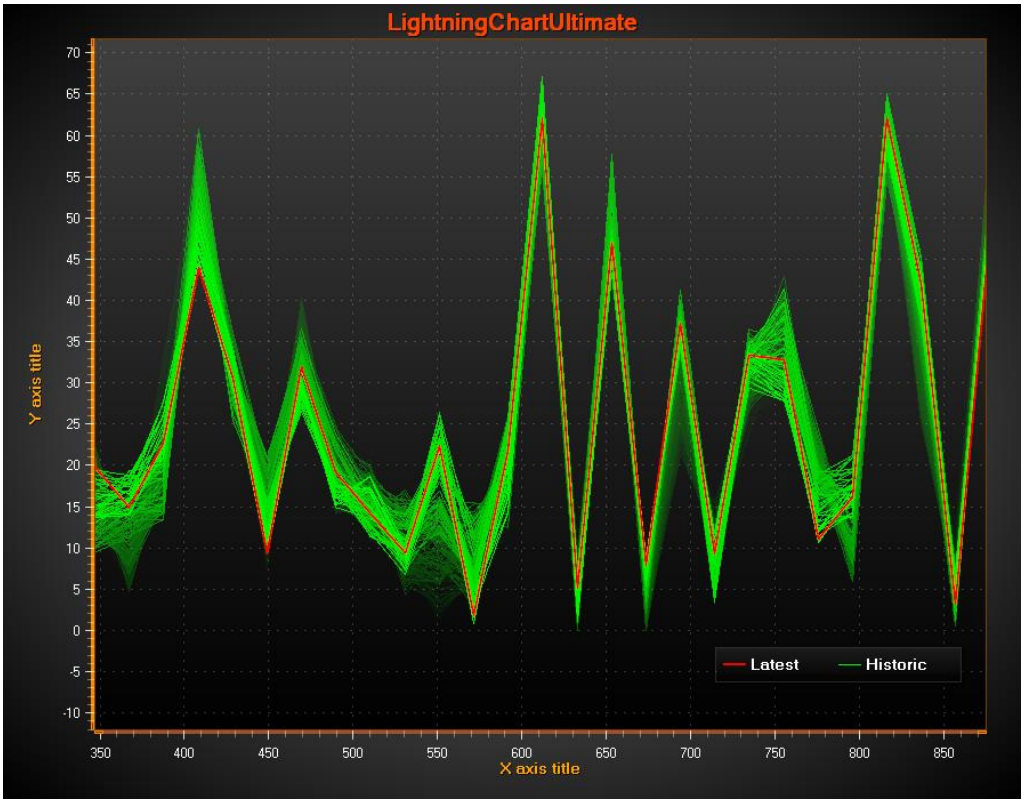


图 6-128. 持续性图层可显示历史轨迹（绿色）。在更新渲染层中的新数据之前调用 MultiplyAlpha 方法，使最早的轨迹消失。

6.38.1 创建图层

PersistentSeriesRenderingLayer 不是 ViewXY 的子属性，并且不能用 Visual Studio 的属性网格来添加。***PersistentSeriesRenderingLayer*** 对象必须在代码中创建。创建方法如下：

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

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

通过提供 ViewXY 对象作为参数，这可绑定在 ViewXY 中的图层。提供相同的 XAxis 对象，该对象与在其上渲染的系列一起使用。

多个图层将按照图表的创建顺序渲染。

6.38.2 清除图层

layer.Clear（）可清除图层，并用 ARGB=（0,255,255,255）对颜色初始化。

layer.Clear（Color color）用给定的颜色来清除图层。在大多数情况下，设置与在背景中使用的同样颜色非常有用，但要设置其 A = 0。 若背景为黑色的，则使用 layer.Clear（Color.FromArgb（0,0,0,0））;

6.38.3 调整图层透明度

MultiplyAlpha（value）可让图层更透明或不透明。将图层中每个像素单独相乘。倍增操作会分别对图层中的每一个像素起作用。

给定值 < 1, 透明度增加（衰减图层）

给定值> 1, 不透明度增加（让图层的可见性更加）

值为 1 时无效。

例如， ***MultiplyAlpha（0.8）***即设置透明度为当前透明度的 80%。***MultiplyAlpha（2）***则调整其为 200%。

6.38.4 渲染数据到图层中

用 ***PointLineSeries***、***SampleDataSeries***、***FreeformPointLineSeries***、***HighLowSeries*** 或 ***AreaSeries*** 中任一个对象可将数据渲染到图层中。这些系列可以是添加到 ***ViewXY.PointLineSeries***、***ViewXY.SampleDataSeries***、***ViewXY.FreeformPointLineSeries***、***ViewXY.HighLowSeries*** 或 ***ViewXY.AreaSeries*** 集中的。还未添加到这些集中的临时系列也可以使用。以常规方式填充数据至系列中（***PointLineSeries*** 可参阅第 6.6.4 章节，***SampleDataSeries*** 可参阅第 6.8.2 章节，***FreeformPointLineSeries*** 可参阅第 6.9 章节，***HighLowSeries*** 可参阅第 6.16.4 章节，***AreaSeries*** 可参阅第 6.17.1 章节） 。

layer.RenderSeries（PointLineSeriesBase series）: 在图层上渲染一个系列。

layer.RenderSeries（List<PointLineSeriesBase> seriesList）: 在图层上渲染所有给定的系列。这比分别为每个系列调用 ***layer.RenderSeries（PointLineSeriesBase series）*** 更有效。

注意! 所有给定的系列将在图层上渲染，即便其 Visible 设置为 False。

注意! 与该系列一起使用的 X 轴必须与提供给 PersistentSeriesRenderingLayer 构造函数的 X 轴相同。否则，将跳过该系列对象。

注意！RenderSeries 用以渲染至图层中。在普通的系列之前，图层自己会先渲染 （ PointLineSeries, SampleDataSeries, FreeformPointLineSeries, HighLowSeries, AreaSeries）.

6.38.5 排列图层

调用 ***layer.Dispose（）*** 可排列图层，并防止图层和图表一起渲染。

6.38.6 消除图层中数据的锯齿

设置 ***layer.AntiAliasing*** 为 ***True，*** 可在图表渲染阶段消除数据锯齿。如果硬件不支持它，它也支持反锯齿。

6.38.7 获得图层列表

ViewXY.GetPersistentSeriesRenderingLayers （ ）返回所有创建的图层列表，包括 ***PersistentSeriesRenderingIntensityLayers.***

6.38.8 需要注意的一些图层限制

由于其特殊的渲染技术，请记住这些限制：

- X 轴的 ***ScrollMode*** 必须设置为 ***None***. 这种方法不可能实现 X 轴的实时滚动。

- 缩放、平移、轴调整以及图表调整大小都将导致图像与轴的范围不同步。在使用持久绘图或应用程序逻辑时，应该禁用这些特性，以便清除图层并临时为新层渲染重新创建之前的线条系列（有用于轴范围更改和大小调整的事件处理程序）。

- 改变图表大小会清除图层，并从 Windows 桌面锁状态重新开始。

- 仅在图层上渲染的系列中不支持鼠标交互

- EMF/WMF/SVG SVG 导出格式，以矢量格式复制到剪贴板以及以矢量格式打印不支持图层。只支持栅格格式。

6.39 持续系列渲染强度图层

演示示例： Intensity persistent layer, signal

PersistentSeriesRenderingIntensityLayer 可以将轨迹聚集到一个图层中，并根据每个像素的点击次数来着色。使用一定值域范围的调色板可以进行着色。轨迹可以采用 ***PersistentSeriesRenderingLayer*** 中同样的系列类型（参阅第 **Error! Reference source not found.**章节）来处理。他们非常相似，主要的区别在于着色。当使用第二次渲染调用在像素位置再次渲染轨迹时，轨迹的强度会增加，从而增加其在值域调色板中的值。

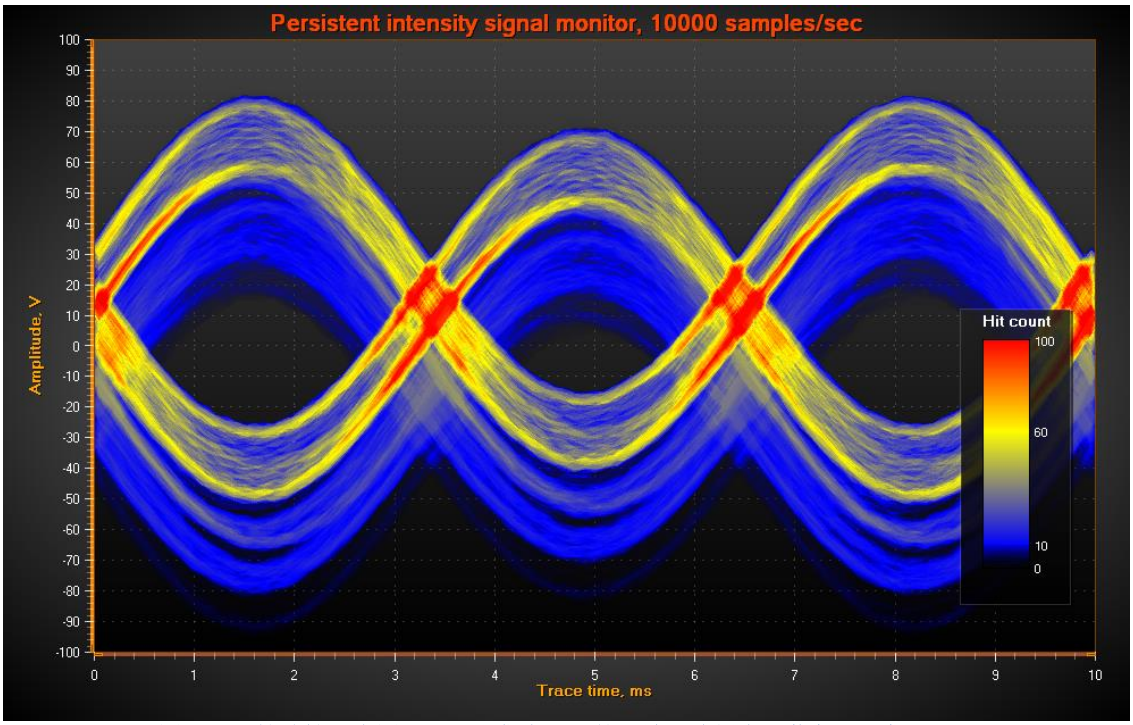


图 6-129. 持续性强度图层凸显出集中活跃的区域，此例中以黄色和红色显示。

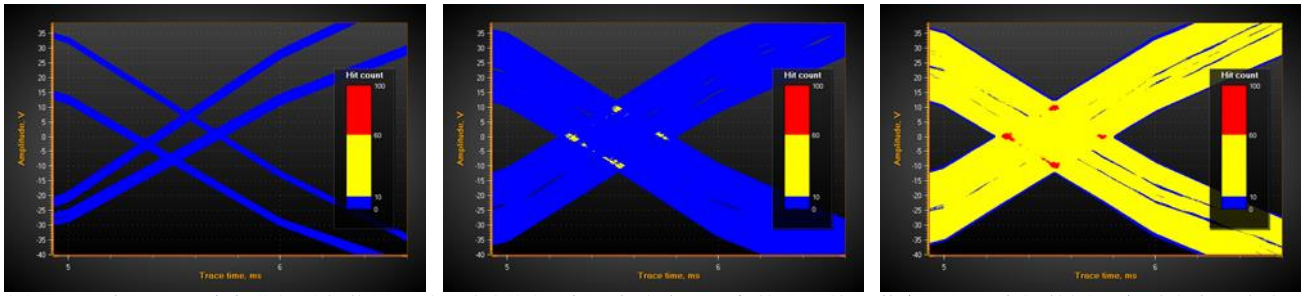


图 6-130. 在同一区域内进行重复信号跟踪。左侧图层中只有渲染了几条轨迹，均以蓝色显示。中间的图层中则渲染了许多条轨迹，但多数在不同的坐标中。在轨迹的交集处，点击次数超过了黄色阈值调色板中定义的 10 个轨迹数。在最右边的图片中，总计渲染了犹数百条轨迹，交集处开始超过给红色定义的阈值。

6.39.1 创建图层

PersistentSeriesRenderingIntensityLayer 不是 ViewXY 的子属性，并且不能用 Visual Studio 的属性网格来添加。***PersistentSeriesRenderingIntensityLayer*** 对象必须在代码中创建。

创建方法如下：

```
using LightningChartLib.WPF.Charting.Views.ViewXY;

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

6.39.2 清除图层

layer.Clear（） 清除图层，并重置计数器。

6.39.3 更改调色板颜色

在图层的 ***ValueRangePalette*** 属性中可定义调色板样式与色阶。Define the palette type and steps in ***ValueRangePalette*** property of the layer.设置 ***ValueRangePalette.Type = Gradient*** 可设定渐变着色，设置 ***ValueRangePalette.Type = Uniform*** 可用离散的色阶来渲染图层。

6.39.4 调整新轨迹强度效应和轨迹衰减情况

用 ***NewTraceIntensity*** 属性来控制使用 RenderSeries 调用渲染的新轨迹的强度效应有多强。典型值范围为 1…100, 这取决于设置颜色范围，以填充轨迹的速度有多快。

用 ***HistoryIntensityFactor*** 来调整之前轨迹的衰减速度。典型值的范围是 0.5 – 0.99。

注意,设置 ***HistoryIntensityFactor*** 本身不会更新图层，只有等到下次调用 ***RenderSeries*** 才行。

6.39.5 渲染数据至图层中

用 ***RenderSeries*** 方法，可将 ***PointLineSeries***、***FreeformPointLineSeries***、***SampleDataSeries***、***HighLowSeries*** 或 ***AreaSeries*** 渲染至图层。

layer.RenderSeries（PointLineSeriesBase series）：渲染图层上的一个系列。

layer.RenderSeries（List<PointLineSeriesBase> seriesList）： 渲染图层上所有给定的系列。但是对于 ***layer.RenderSeries（PointLineSeriesBase series）*** 没有性能提升。

当将数据更新至图层中后，可对新轨迹使用 ***NewTraceIntensity*** 。同事用可令之前的轨迹数据衰减。 ***layer.RenderSeries（List<PointLineSeriesBase> seriesList）*** 则在每个系列对象之后衰减之前的轨迹。

6.39.6 排列图层

用 ***layer.Dispose（）*** 来排列图层，并防止图层与图表一起渲染。

6.39.7 图层中消除数据锯齿

设置 ***layer.AntiAliasing*** 为 ***True***，可在图表渲染阶段消除数据锯齿。如果硬件不支持它，它也支持反锯齿。

6.39.8 获得图层列表

ViewXY.GetPersistentSeriesRenderingLayers （）返回所有创建的图层列表，包括 ***PersistentSeriesRenderingLayers***。

6.40 自定义控件 - 缩放栏 （Zoom bar）

缩放条是一个自定义的 XY 图表，可用于获取整个数据集的概览并将图表缩放到特定区域。 创建缩放栏时，它将引用的图表作为参数。 由于缩放栏是 LightningChart 的一个单独实例，因此它可以放置在与主图表不同的容器中。 需要使用 CustomControls 命名空间才能使用缩放条。

```
using Arction.Wpf.Charting.CustomControls;

// Creating a LightningChart object, then adding a Zoom bar referring to it.
LightningChart _chart = new LightningChart();
mainGrid.Children.Add(_chart);

zoomBarGrid.Children.Add(new ZoomBar(ref _chart));
```

缩放栏自动显示主网格中的所有系列。 但是，包含控制缩放栏行为的所有属性的 ZoomBarOptions 具有允许隐藏特定系列类型的 SeriesToUse 选项。 可以在保持点可见的同时隐藏线，反之亦然。 默认情况下，缩放栏中的点是隐藏的，只有线是可见的。 请注意，如果线或点在主图表中不可见，则它们也无法显示在缩放栏中。

```
// Hiding all PointLineSeries lines in the Zoom bar chart.
zb.ZoomBarOptions.SeriesToUse.PointlineSeries.LineVisible = false;
```

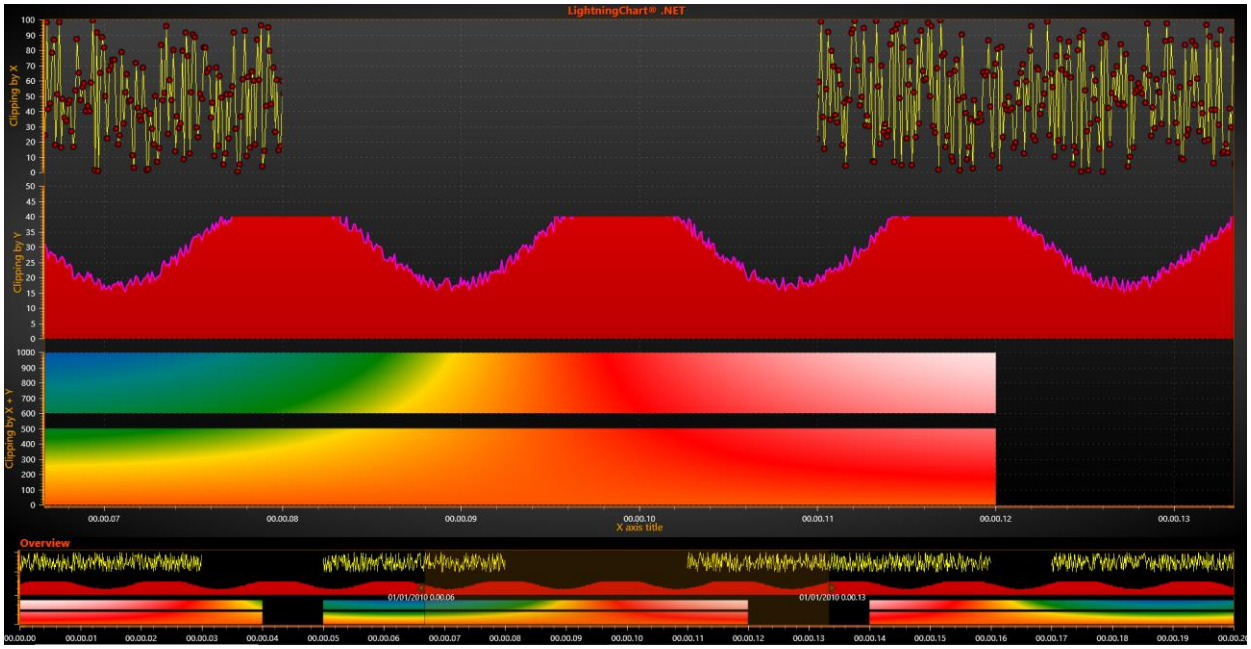


图 6-135. 在主图表下方添加了缩放条，提供了整个数据的概览。 点线系列的数据点在主图表中可见，但对于缩放栏中的相应系列则不可见。.

在不断添加新数据的实时图表中，缩放栏无法自动更新。 在这些情况下，应该使用添加的数据点作为参数来调用特定于系列的添加数据方法，例如 AddDataToPointLineSeries() 或 AddDataToHighLowSeries()。

```
// Updating series in Zoom bar. The first parameter is the series index.
zoomBar.AddDataToPointLineSeries(0, seriesPointArray);
```

6.40 自定义控件 Violin plot

小提琴图是一种自定义 XY 图表，它使用密度曲线描绘一个或多个组的数值数据分布。 它是 LightningChart 自己的实例，这意味着它需要添加到父容器（例如网格）中。 需要使用 CustomControls 命名空间来创建 Violin 图。

```
using Arction.Wpf.Charting.CustomControls;

ViolinPlot _violin = new ViolinPlot();
containerGrid.Children.Add(_violin);
```

将数据添加到小提琴图是通过 AddGroupData() 方法完成的。 该方法采用多个有关其大小和位置的参数，例如最小值、最大值和宽度，以及样式选项，例如颜色和标签文本。 该方法的最后一个参数是作为 PointDouble2D 数组的实际数据点。 通过多次调用 AddGroupData() 可以将几把小提琴添加到同一个绘图中。

```
// Adding data to violin plot.
_Violin.AddGroupData(45, 95, 1, 0.5, "Group A", Colors.Yellow, "A",
pointArray1);
_Violin.AddGroupData(54, 79, 2, 0.5, "Group B", Colors.Magenta, "B",
pointArray2);
_Violin.AddGroupData(56, 97, 3, 0.5, "Group C", Colors.Orange, "C",
pointArray3);
```

可以使用 SetXaxisTitle() 和 SetYaxisTitle() 方法更改小提琴图的轴标题。 Ypadding() 可用于设置小提琴和图表边缘之间留有多少垂直空白空间。

Violin 图所基于的常规 LightningChart 对象可以通过 GetInnerChart() 方法访问。 这允许修改例如构建小提琴的多边形对象。

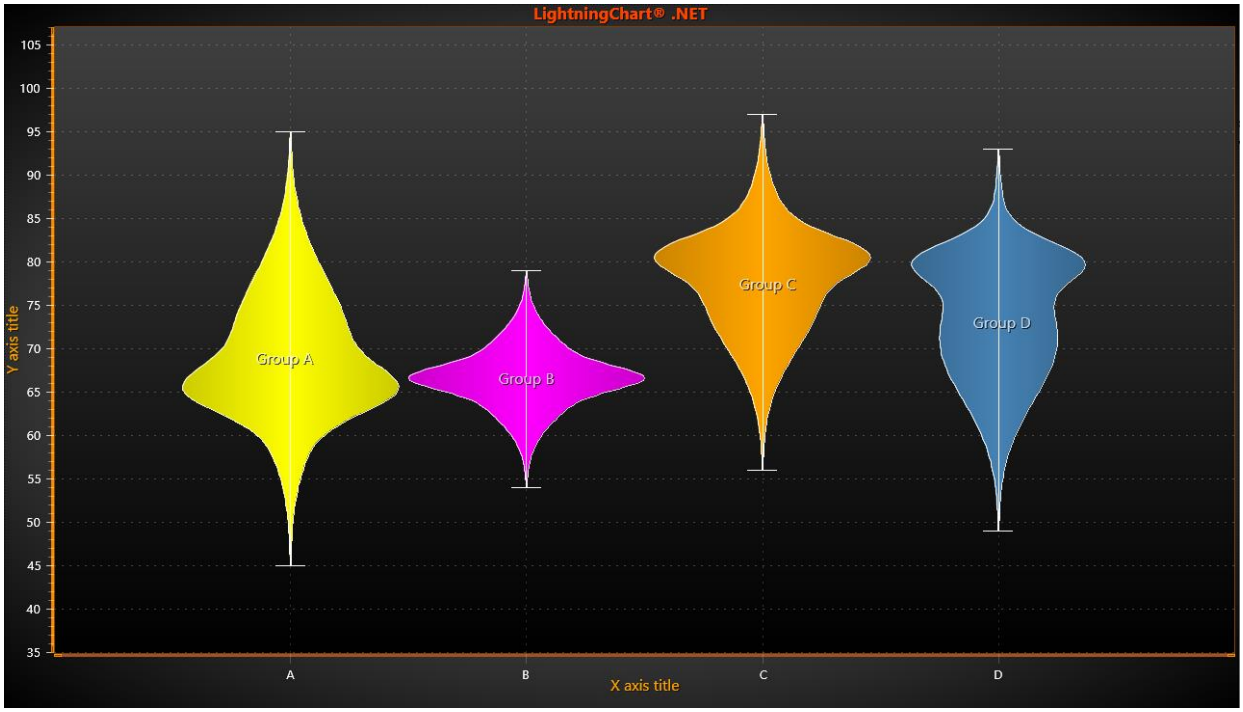


图 6-136. 创建完成的小提琴图。 通过 AddDataGroup() 方法添加了个小提琴。

6.41 Image Layer

ImageLayer 是一个非常大的图像层，可以逐渐用较小的图像填充。较小图像的累积是在后台线程中完成的，从而保持图表的 UI 线程响应迅速（例如，可以在加载图像时进行缩放）。虽然从许多较小的位图组合 ImageLayer 需要大量内存，但图表本身仍然非常具有交互性且速度很快。所有 ViewXY 系列都可以在 ImageLayer 之上渲染。此外，可以创建多个 ImageLayer，并且可以独立打开/关闭每个层的可见性。

将所有源图像添加到图层后（由 `layer.EndAddingImages()` 方法调用指示），可以在主图层之上创建多个子图层。每个子图层代表缩放级别，图层中的像素数不断减少（近似减少因子为 4）。这允许在应用程序中与所有缩放级别进行快速交互（工作方式类似于地理地图的图块）。`EndAddingImages()` 方法的输入参数定义应创建多少个子图层/缩放级别。当所有完整图层都可见时，将显示细节数量最少的子图层。这允许与图表快速交互（例如平移），同时提供对 `ImageLayer` 的良好概览。当图表放大时，将显示更详细的子图层，直到达到主图层（0 级）。如果有关于每个子图像位置和大小的信息（以 XY 轴为单位），则可以创建任意大小的拼贴画。

为了获得最佳性能和最少的内存使用量，建议以 `Indexed8` 颜色空间格式向 `ImageLayer` 提供图像。否则，内部图像将转换为 32 位 RGBA 格式（这需要约 3 倍的内存）。

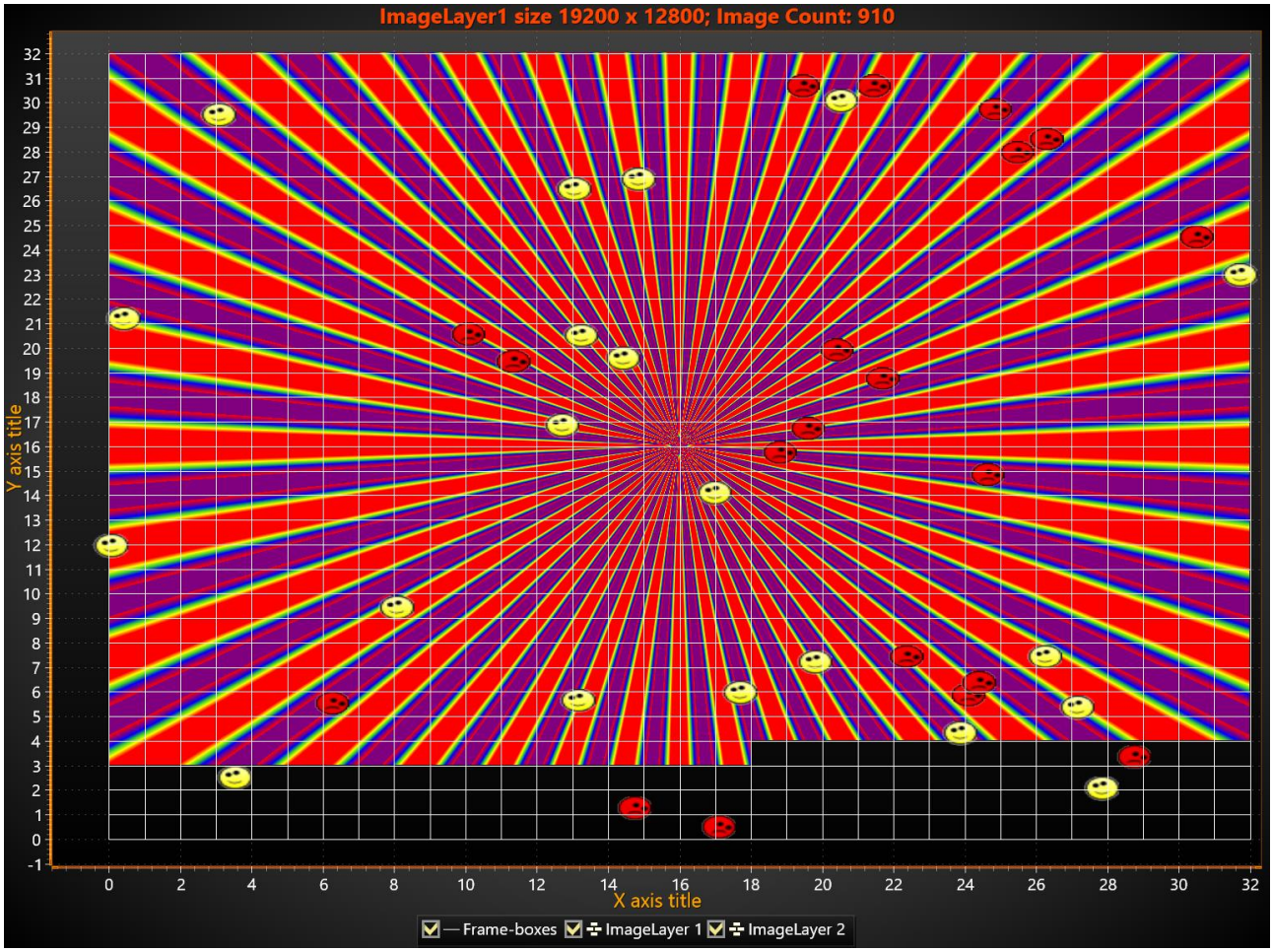


图 6 136. 两个 ImageLayers 正在使用中，其中一个仍在加载中

6.42 Custom controls - Violin plot

小提琴图是一种自定义 XY 图表，使用密度曲线描绘一个或多个组的数字数据分布。它是 LightningChart 的一个实例，这意味着需要将其添加到父容器（例如网格）中。需要使用 CustomControls 命名空间才能创建小提琴图。

```
using LightningChartLib.WPF.Charting.CustomControls;

ViolinPlot _violin = new ViolinPlot();
containerGrid.Children.Add(_violin);
```

通过 AddGroupData() 方法向小提琴图添加数据。该方法采用几个有关其大小和位置的参数，例如最小值、最大值和宽度，以及样式选项，例如颜色和标签文本。该方法的最后一个参数是 PointDouble2D 数组形式的实际数据点。通过多次调用 AddGroupData()，可以将多个小提琴添加到同一个图中。

```
// Adding data to violin plot.
_Violin.AddGroupData(45, 95, 1, 0.5, "Group A", Colors.Yellow, "A",
pointArray1);
_Violin.AddGroupData(54, 79, 2, 0.5, "Group B", Colors.Magenta, "B",
pointArray2);
_Violin.AddGroupData(56, 97, 3, 0.5, "Group C", Colors.Orange, "C",
pointArray3);
```

可以使用 SetXaxisTitle() 和 SetYaxisTitle() 方法更改小提琴图的轴标题。Ypadding() 可用于设置小提琴和图表边缘之间留出多少垂直空白空间。

小提琴图所基于的常规 LightningChart 对象可通过 GetInnerChart() 方法访问。这允许修改例如小提琴所构建的多边形对象

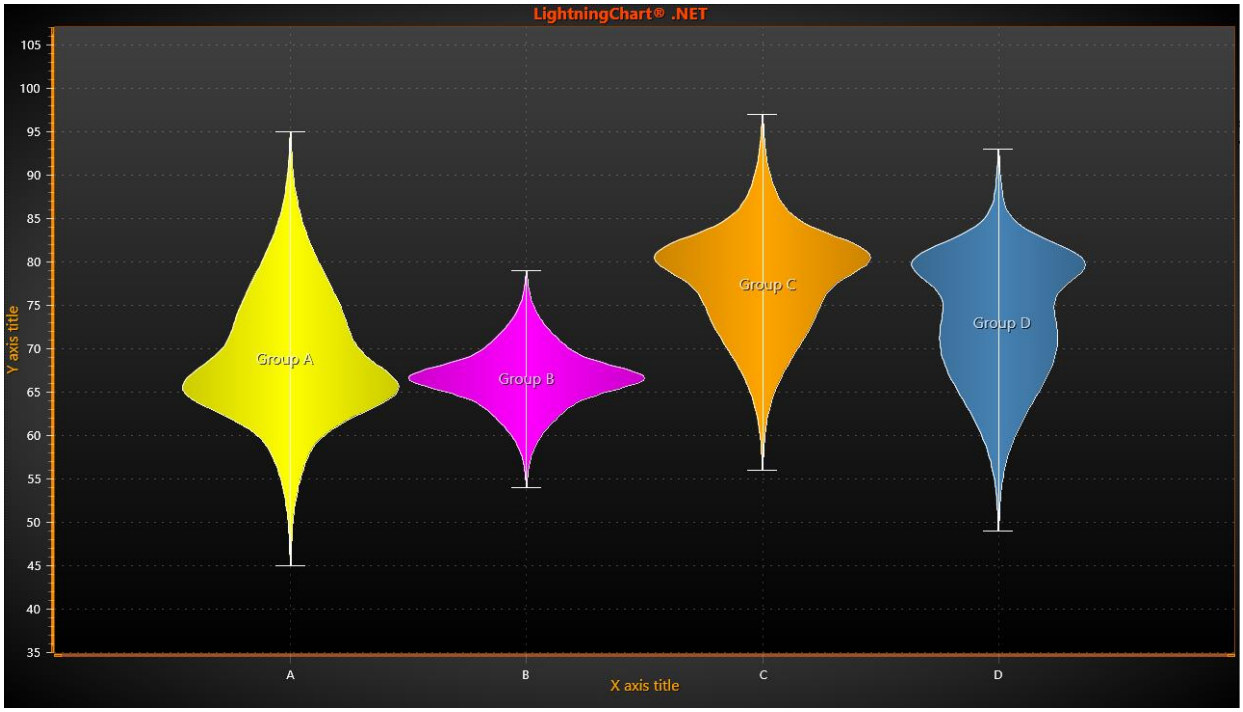


图 6 137. 小提琴图已创建。已通过 AddDataGroup() 方法添加四把小提琴。

7. View3D

View3D 可以在 3D 空间中实现数据可视化。3D 模型可用各种方法来缩放、旋转、点亮。**不同的系列类型可以放置到相同的 3D 视图中，以实现组合可视化。**

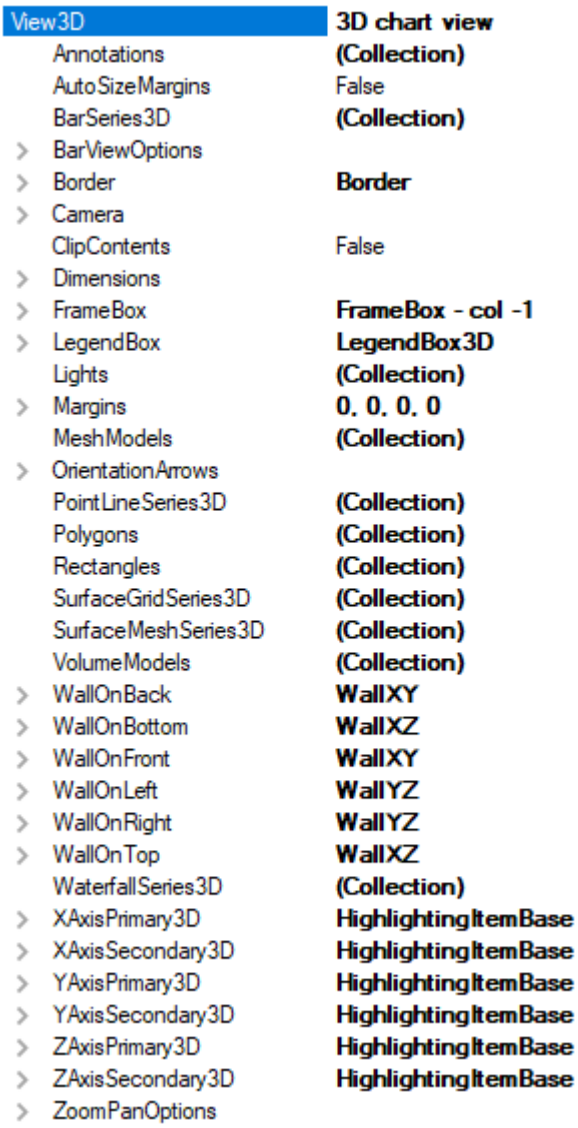


图 7-1. View3D 对象主树

7.1 3D 模型及尺寸

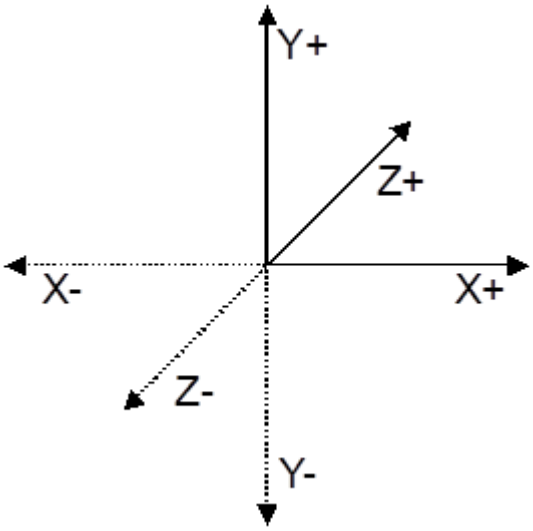


图 7-2. 3D 模型正负方向。

在 3D 空间中心创建 3D 模型。维度量级定义了三维空间中模型箱的大小。用该维度箱可以定义墙和轴的大小。用 ***Dimensions*** 属性可设置每个维度的量级。

当没有定义照相机旋转时，正 X 方向朝右，正 Y 方向朝上，正 Z 方向朝向屏幕内。

7.1.1 World coordinates 全局坐标

一些 3D 对象采用的是 "*全局坐标*", 而非轴值。例如，光的定位方式是独立于轴的范围。全局坐标还可以称为"*3D 模型空间坐标*".

原点 [0,0,0] 位于模型的中心。实际的 3D 模型空间范围开始于 [-Dimensions.X/2 至 Dimensions.X/2], [-Dimensions.Y/2 至 Dimensions.Y/2] 和[-Dimensions.Z/2 至 Dimensions.Z/2].

LightningChart 提供有专门的方法可以在系列值、轴值、世界坐标和屏幕坐标间进行值得转换。更多详情可浏览演示应用程序示例及帮助文档。

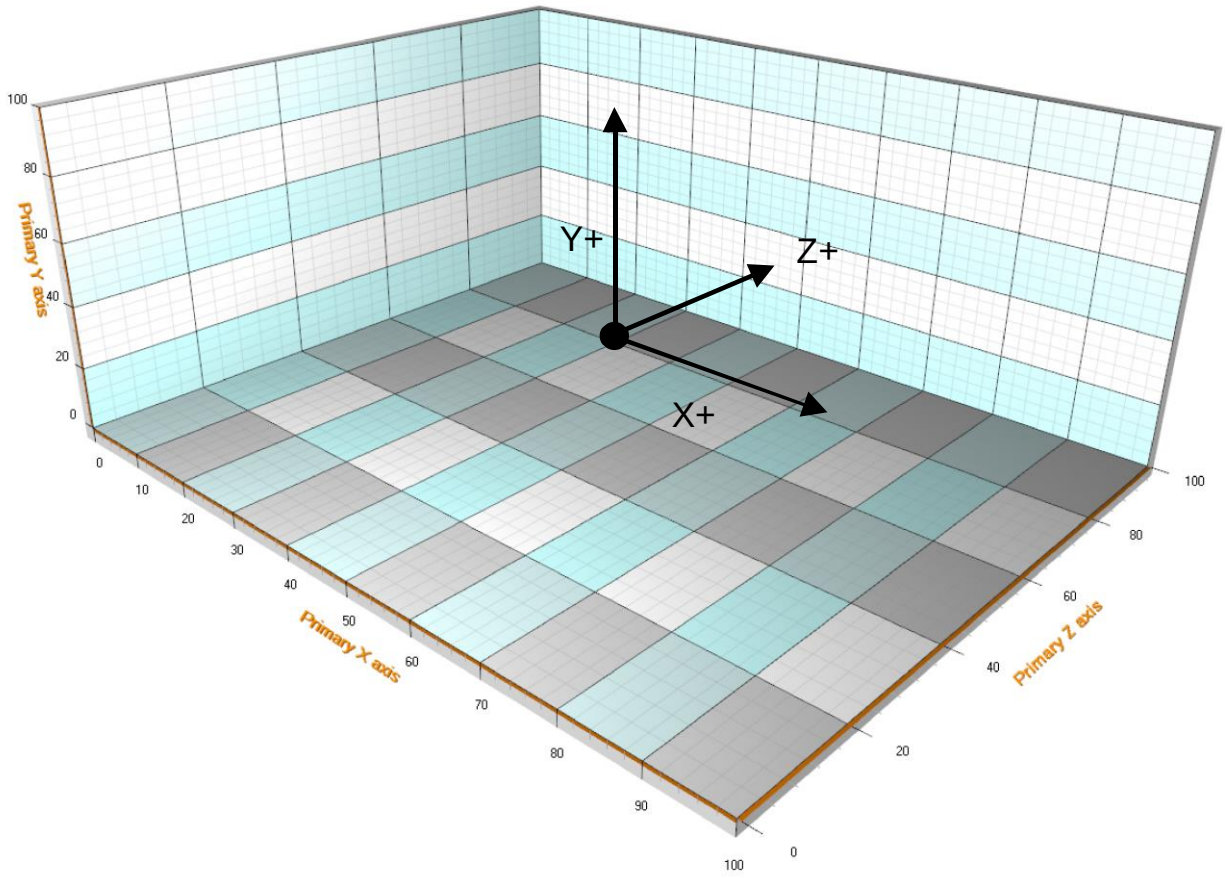


图 7-3. 3D 试图设置示例。维度设置为 X=100, Y=40, Z=80. 可见墙：左面、底面、后面。采用透视摄像头视角。

7.2 Walls 墙

墙 （***WallOnFront***, ***WallOnBack***, ***WallOnTop***, ***WallOnBottom***, ***WallOnLeft***, ***WallOnRight***）用于表示轴网格和网格条，并为轴提供基础。默认情况下，底部、左、右、前、后的墙都可见，其 ***AutoHide*** 属性设置为 true。当旋转视图时，阻隔墙被暂时隐藏，这样就不会阻挡住图表内容的视图。若要让墙可见，则设置 ***Visible = true*** 且 ***AutoHide = false***。

使用 XGridAxis，YGridAxis，ZGridAxis，GridStripColorX，GridStripColorY，GridStripColorZ 和 GridStrips 属性可从中选择应用网格的轴，并修改网格的颜色网格条。 可用的属性取决于墙的方向。 隐藏实心墙时 FullTransparent 属性允许仅显示网格。 请注意，即使启用了 FullTransparent，网格仍然遵循墙的可见性和自动隐藏属性。

7.3 FrameBox （框架箱）

用一个简化了的 3D 箱外观可替代墙。为每面墙设置 Visible = false，接着设置 ***FrameBox.Style = AllEdges***。用 ***FrameBox.LineColor*** 可设置颜色或框架。

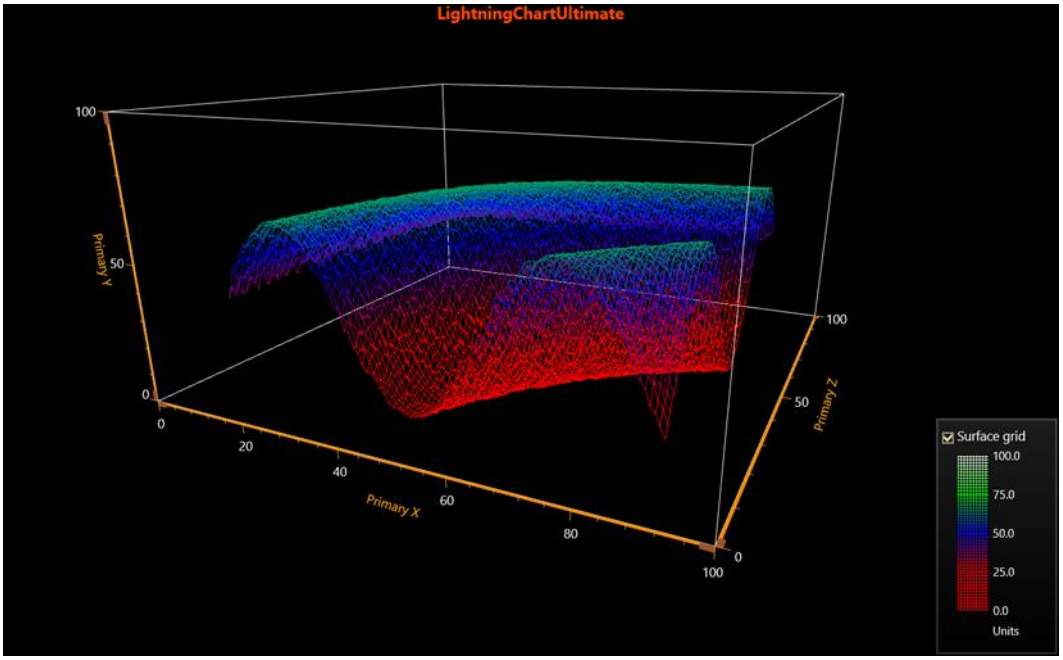


图 7-4. FrameBox 可视，隐藏掉墙

7.4 摄像头

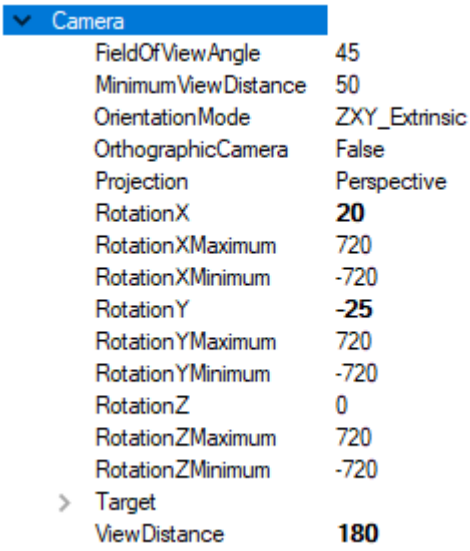


图 7-5. Camera 属性.

摄像头类型、位置、距离和朝向一起决定了 3D 的视角。用 ***RotationX***, ***RotationY***, ***RotationZ*** 和 ***ViewDistance*** 可设置摄像头在 3D 模型空间中的位置。通过设置 ***Target*** 属性，可以将摄像头对准优先选定的方向。

用 ***Projection*** 属性可选择投影类型。

- ***Perspective,***显示一个具有真实感的投影。
- ***Orthographic***, 此为在科学和工程应用中使用的投影类型。比起 ***OrthographicLegacy*** 更推荐用此选择。

- ***OrthographicLegacy***，（相当于在 LightningChart v.8.3 或更早的版本中设置 OrthoGraphicCamera = True）。相比 Orthographic 而言，这在缩放后渲染更慢 。它维护三维对象的大小，这在如果 3D 对象是在 3D 全局坐标（不是轴值）中定义的情况下，可维持 3D 对象的大小。而且，墙的厚度在缩放时会保持不变。缩放会改变维度尺寸，但不会影响 ***ViewDistance***.

RotationX, RotationY and ***RotationY*** can be limited by setting boundaries 通过 ***RotationXMinimum***, ***RotationXMaximum***、***RotationYMinimum***、***RotationYMaximum***、***RotationZMinimum*** 和 ***RotationZMaximum*** 属性来设置边界可限制 ***RotationX***、***RotationY*** 和 ***RotationY***。

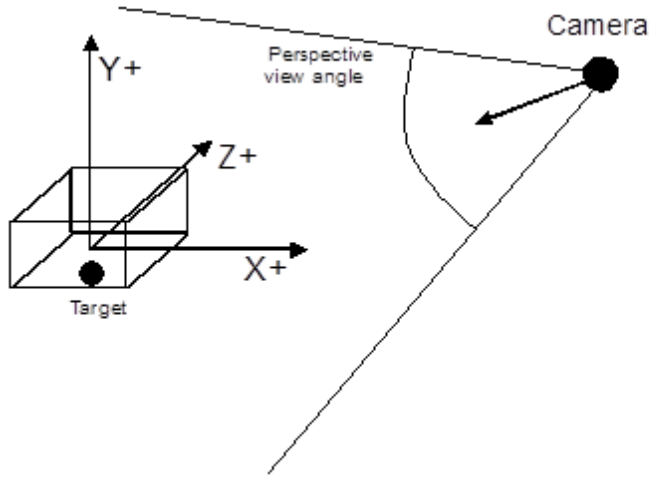


图 7-6.透视摄像头在 3D 空间的样子

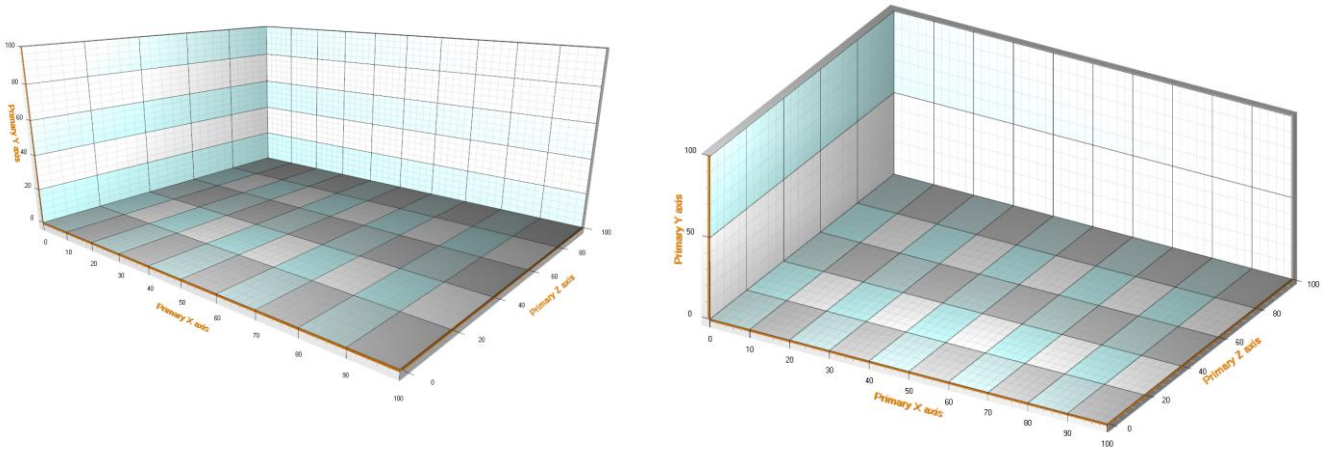


图 7-7. 3D 空间的透视和正交摄像头视角。

缩放后的试图在 Orthographic 和 OrthographicLegacy 中的不同如下：

Orthographic

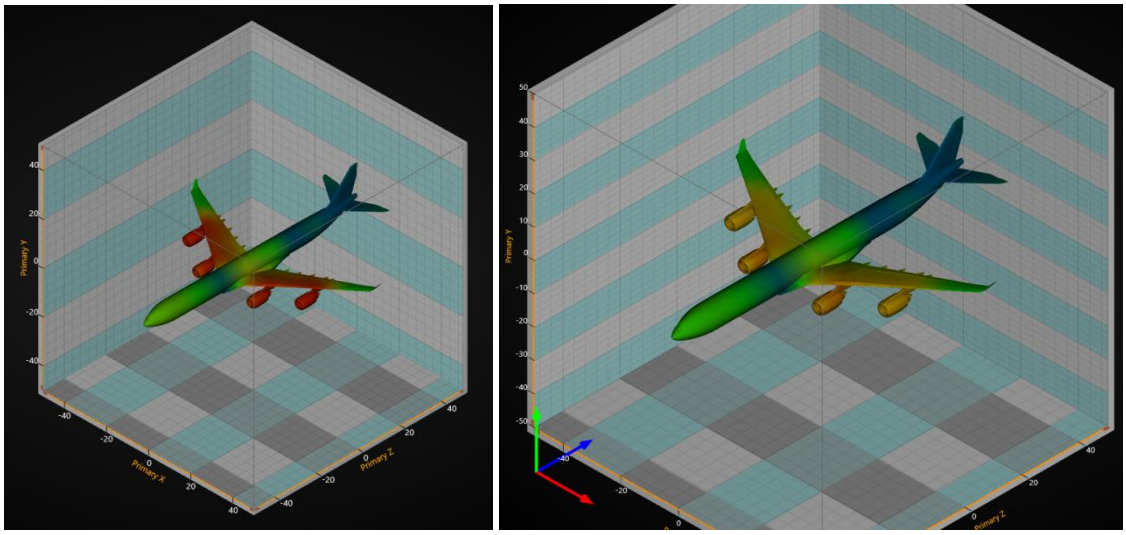


图 7-8. Orthographic 投影类型。左侧无缩放。右侧为放大后。飞机（MeshModel3D）对象的大小与其他对象一起在屏幕上扩大。

OrthographicLegacy

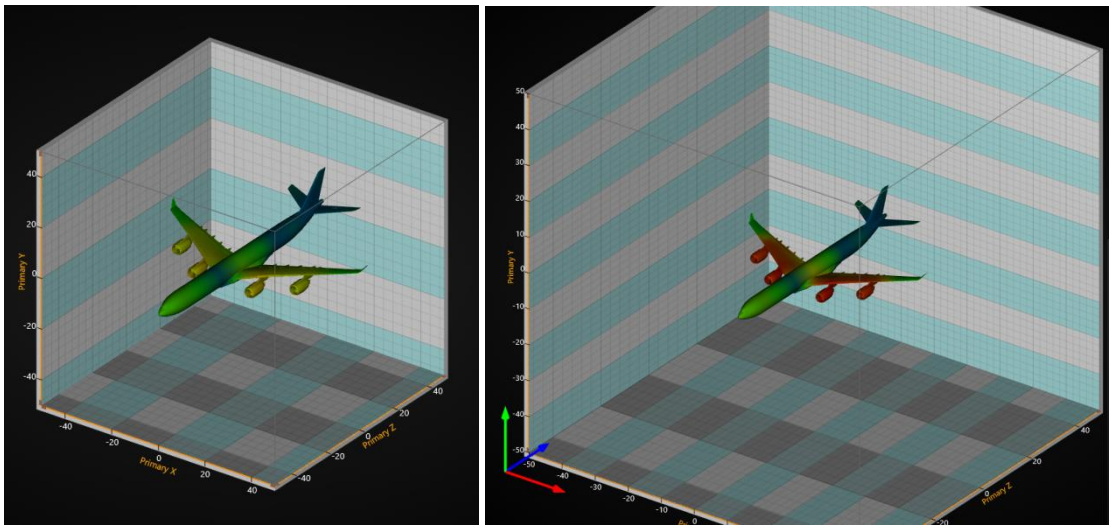


图 7-9. OrthographicLegacy；左侧无缩放。右侧为放大。On the left, unzoomed. On the right, zoomed in. Airplane （MeshModel3D, see 7.14）对象尺寸保持不变，但 3D 尺寸改变了。

7.4.1 预先设定摄像头

演示示例： Cameras and lights

用 ***View3D.Camera*** 的 ***SetPredefinedCamera*** 方法可设置一个预先设定好的摄像头。

```
//设置预定义的摄像头方向
chart.View3D.Camera.SetPredefinedCamera（PredefinedCamera.BackOrthographic）;
```

7.4.2 摄像头方向模式

在 LightningChart v8.4 版本中，增加了一个新的摄像头方向模式与改进的摄像头方向定义。现在被设置为默认方向模式的新模式叫做 ***ZXY_Extrinsic*** （此名称规定了维度轴的计算顺序）。它修复了许多基于旋转的问题，特别是在图表的两极附近（例如图表上方的摄像头）。之前的方向模式 ***XYZ_Mixed*** 仍然可用，但很可能在未来的某个时候不宜再用。通过 ***View3D.Camera.OrientationMode*** 可以获取方向信息。

这种变更夜会对旋转进行修改。采用新的摄像头方向模式，其中一个轴方向（世界单位矢量）被用作鼠标水平旋转轴。这即是摄像头旋绕的轴。当 ***RotationX***、***RotationY*** **或** ***RotationZ*** 属性改变时，轴的测定可自动完成。最靠近摄像头的向上方向的轴选做旋转轴，这样可使旋转在任何情况下都尽可能自然。

通过新的方向和旋转模型可以查看 3D 场景，这在以前是不可能的。

7.5 Lights 光

光可在 3D 模型空间中自由放置在任何地方。若有多条光，则可添加到 ***Lights*** 集合属性中。光有两种不同的类型： ***Directional*** and ***PointOfLight***.

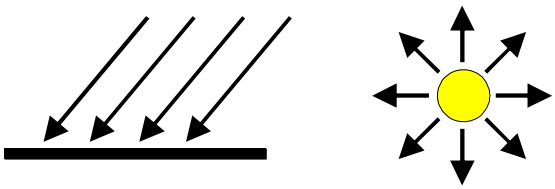


图 7-10. 平行光与光点。

注意! 通过 ***SuppressLighting*** 属性，一些系列类型可以从其表面完全遮盖住光。如果应该正确地照亮该系列，要确认它没有开启。曲面系列具有 ***LightedSurface*** 属性，可选择正确照亮的面一侧。

注意! 将所有光放置在 3D 模型箱内会让墙边缘看上去非常暗，可能会让轴的厚度难以可见。在这种情况下，可调整轴刻度颜色。

7.5.1 平行光

在 ***Directional*** 光中，光线均是平行的，并且光强度不会随着距离的增加而减弱。根据 ***Location*** 和 ***Target*** 属性，光通量可获得方向。***LocationFromCamera*** 属性可采用摄像头的位置作为光源。

7.5.2 光点

在 ***PointOfLight*** 中，光强度会随着距离增加而衰减。用 ***AttenuationConstant***、***AttenuationLinear*** 和 ***AttenuationQuadratic*** 属性可控制光方向上的衰减情况。因为光均等地向各个方向分散，所以这种光类型的 ***Target*** 无关紧要。

7.5.3 光与材质

所有的 3D 对象都有一种 ***Material*** 属性。根据材质可以判断出是如何反射光的。材质的 ***DiffuseColor*** 与光的 ***DiffuseColor*** 起反应。材质的 ***SpecularColor*** 与光的 ***SpecularColor*** 起反应。漫反射的颜色可以理解为一种哑光的基础色，而高光的颜色则是被照亮的表面反射的颜色。采用高度的 ***SpecularPower*** 可以让对象有种金属质感的外观。

曲面系列具有 ***ColorSaturation*** 属性，有效范围为 0…100%。高的值可增强表面填充色，降低阴影效果。

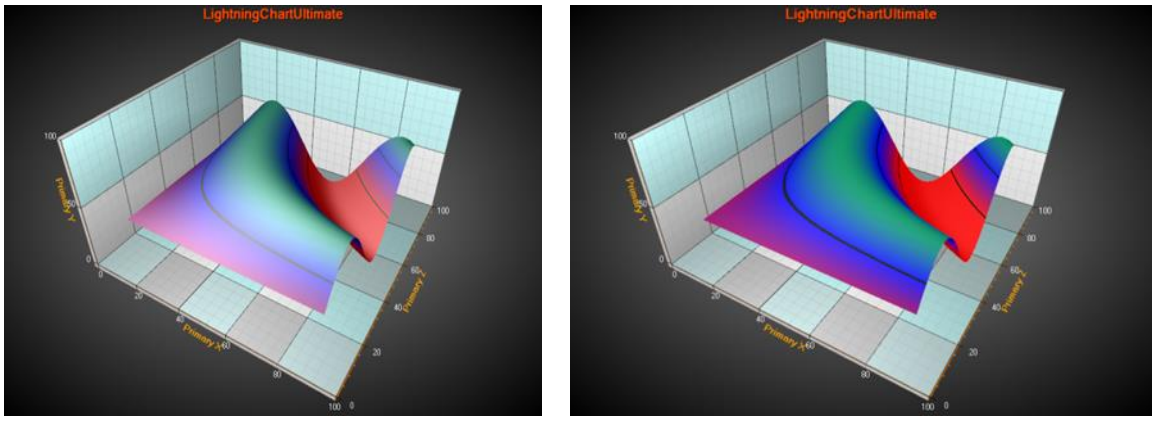


图 7-11. 左侧曲面系列的 ColorSaturation = 50%；而右侧则 ColorSaturation = 85%.

7.5.4 预定义的光照设计

演示示例： Cameras and lights

用 View3D 的 ***SetPredefinedLightingScheme*** 方法可选择一种内置的预定义光照设计。

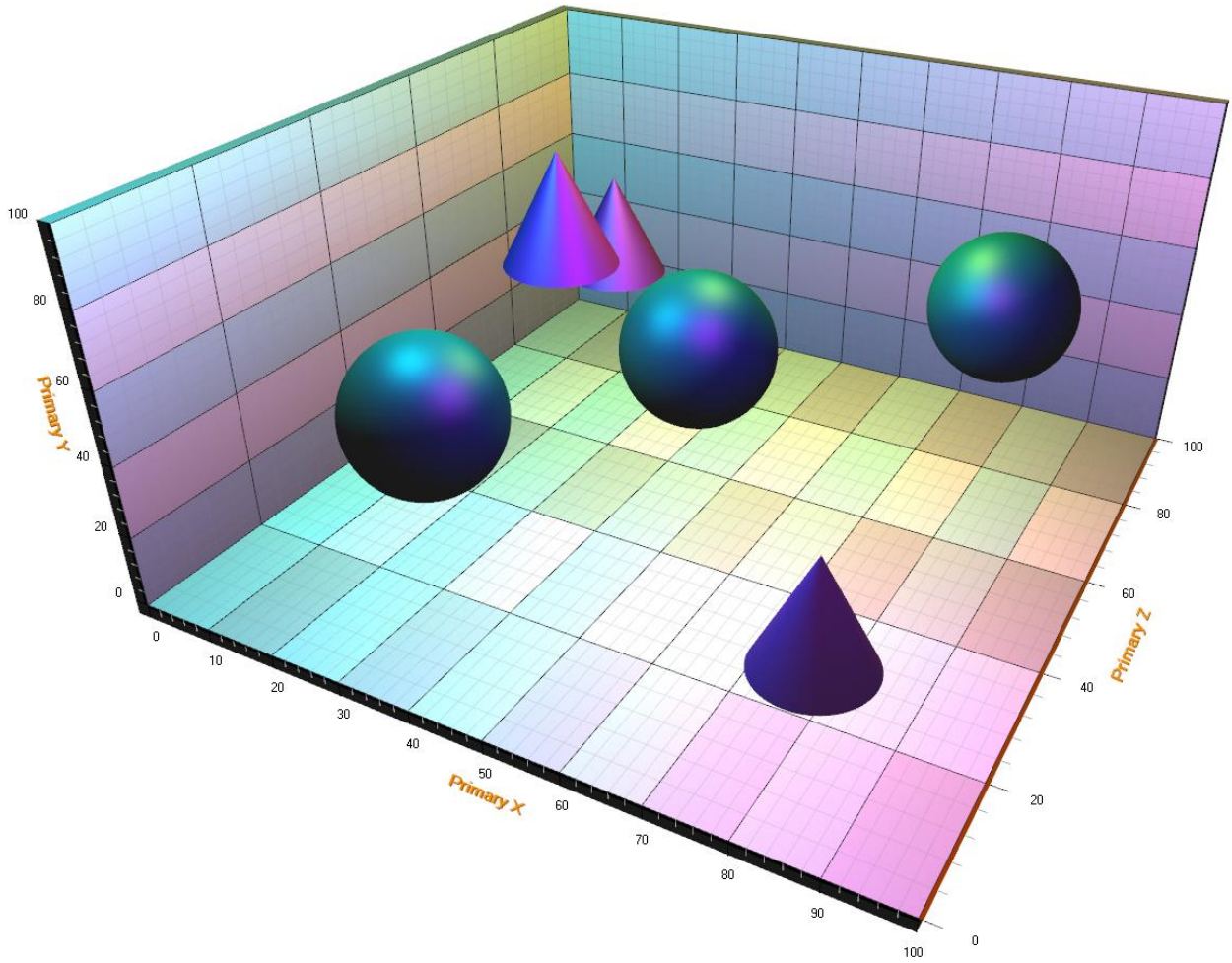


图 7-12. 预定义的‘DiscoCMY’设计。 此设计是由靠近最高上限的三个着有不同颜色的 PointOfLights 组成。

7.6 Axes 轴

每一个维度，都有两条轴：主轴和副轴。即 View3D 视图具有以下轴属性： ***XAxisPrimary3D***、***XAxisSecondary3D***、***YAxisPrimary3D***、 ***YAxisSecondary3D***、***ZAxisPrimary3D*** 和 ***ZAxisSecondary3D***。

通常来说，3D 视图的轴与 ViewXY 视图的轴表现的非常像。二者的许多属性与方法都相似。

7.6.1 Location 位置

在 3D 视图中，轴可以放置于 3D 模型箱角落。利用轴的 ***Location*** 属性可以调节位置。

- X 轴的 ***Location*** 选项有： ***BottomFront***、***BottomBack***、***TopFront*** 和 ***TopBack***.
- Y 轴的 ***Location*** 选项有：***FrontLeft***、***FrontRight***、 ***BackLeft*** 和 ***BackRight***.
- Z 轴的 ***Location*** 选项有：***BottomLeft***、***BottomRight***、***TopLeft*** 和 ***TopRight***.

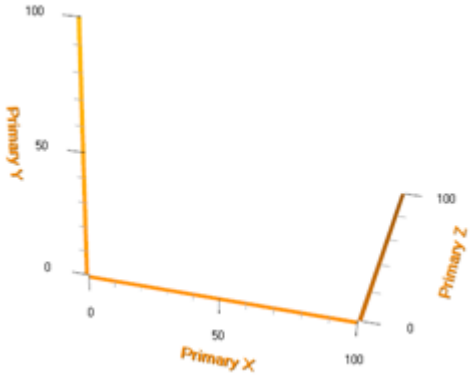


图 7-134. 默认的轴位置设置， XAxisPrimary 在 BottomFront, YAxisPrimary 在 FrontLeft ， ZAxisPrimary 在 BottomRight 内。

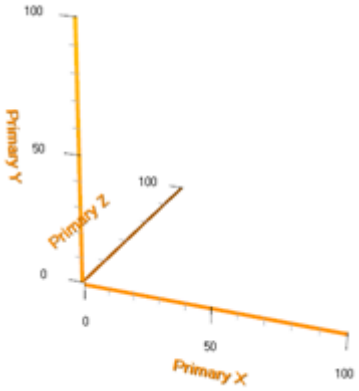


图 7-143. ZAxisPrimary 位置设置为 BottomLeft.

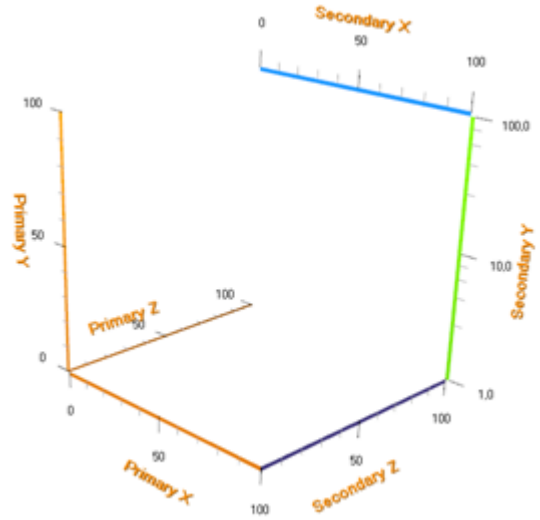


图 7-15. 副轴设置为可见，其位置与颜色可任意设置。副 Y 轴的 ScaleType 设置为 Logarithmic .

7.6.2 轴向（Orientation）

每个轴可以在两个平面上导向。这会影响到轴刻度和值标签的位置与方向。

- X 轴：XY 和 XZ 平面
- Y 轴：XY 和 YZ 平面
- Z 轴：XZ 和 YZ 平面

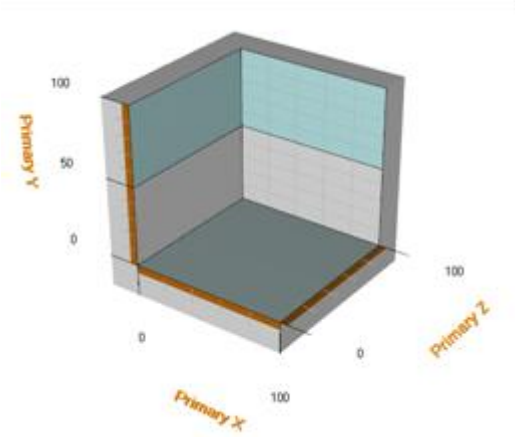


图 7-16. X 轴导向设置为 XY, Y 轴导向设置为 XY, Z 轴导向设置为 XZ.

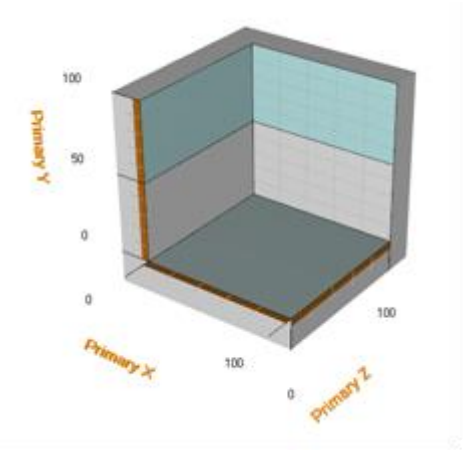


图 7-17. Y 轴导向不变，但 X 轴导向变更为 XZ，同时 Z 轴导向变更为 ZY 面.

7.6.3 CornerAlignment

3D 模型箱角的轴对齐可以通过 ***CornerAlignment*** 属性进行更改。用 ***MajorDivTickStyle*** 和 ***MinorDivTickStyle Alignment*** 属性可控制文本对齐。

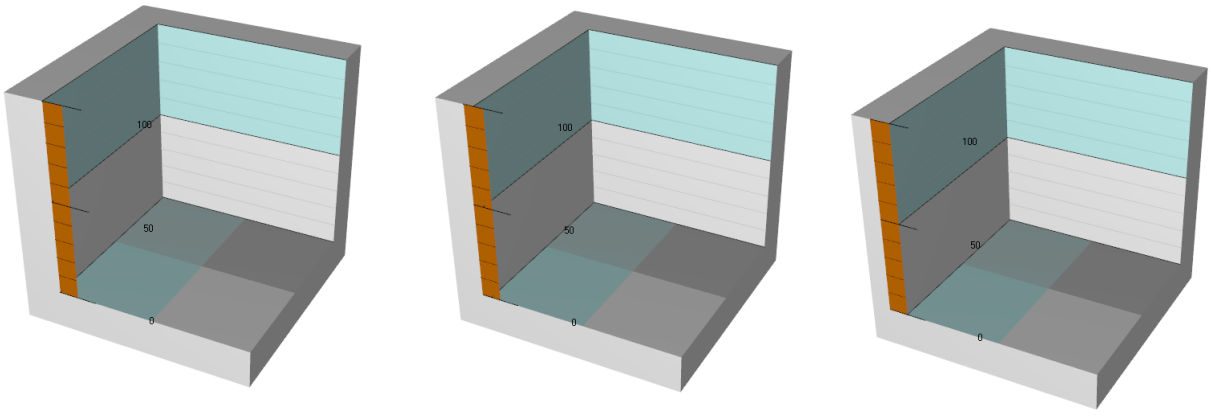


图 7-18. 本例中只有 Y 轴可见。第一张图： Y 轴的 CornerAlignment 设置为 Inside； 在 MajorDivTickStyle 和 MinorDivTickStyle 中的对齐属性设置为 Near；第二张图： CornerAlignment 设置为 AtCorner；第三张图：CornerAlignment 设置为 Outside.

7.7 Margins （图边距）

从 LightningChart v.8.4 版本开始， View3D 视图支持图边距。与 ViewXY 视图类似的是, 当 ***AutoAdjustMargins*** 设置为 ***true*** 后，可以对图形尺寸进行调节来腾出足够空间来放置所有轴以及图表标题。如果设置其为 ***disabled***, ***View3D.Margins*** 属性适用于手动设置图边距。默认情况下，***AutoAdjustMargins*** 设置为 ***false***。

View3D.MarginsChanged 事件可以设置为在图边距变更（例如调整大小）时触发。

图边距之外的视图内容会被自动裁剪掉。除了图表标题、注释和图例框外，其他内容都会被剪切，因为它们的位置是在屏幕坐标中定义的，所以它们夜可以自由地放置在图边距上。另外可以绘制一个 1 像素宽的矩形边框（***Border***）来显示图边距的位置所在。默认情况下，边框在 View3D 视图中不可见。通过 ***Border.Color*** 可以修改此矩形边框的颜色。

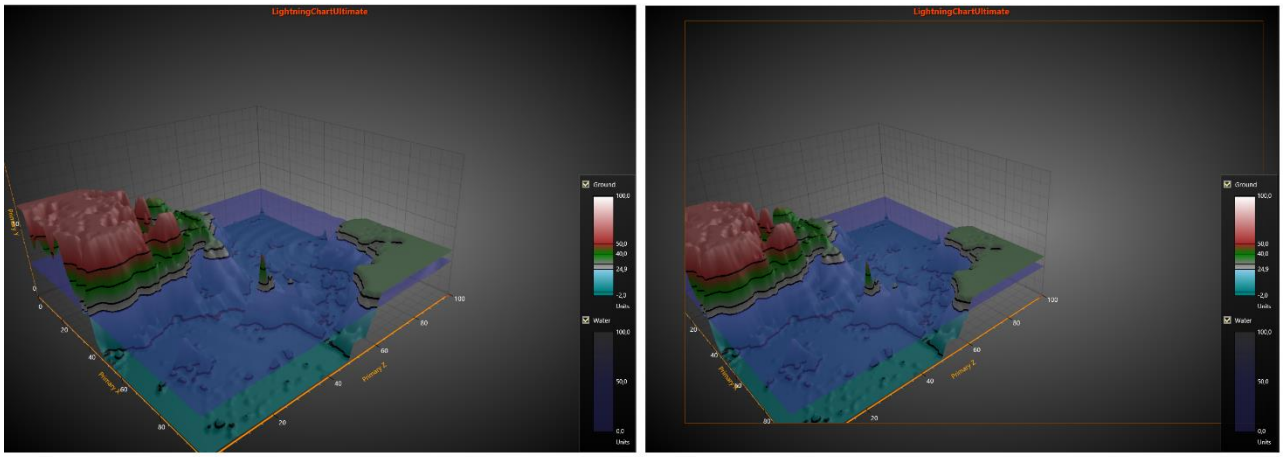


图 7-19. 左侧图形没有图边距（所有图边距设置为 0）。右侧设置有图边距，并且外部的内容都被裁剪掉了。Border.Visible 设置为 True，标记出了视图的图边距的位置。

7.8 3D 系列，全视图

View3D 视图的系列可以以不同的方法和格式实现数据可视化。所有系列都连接到轴值范围。对于每个维度来说，可以选择将系列连接到主轴或副轴；具体可以用 ***XAxisBinding***、***YAxisBinding*** 和 ***ZAxisBinding*** 属性进行控制。

7.9 PointLineSeries3D

演示示例： Scatter points; Point lines; Points tracking; Point cloud; Parallel coordinates chart; Multi-colored 3D point-line

PointLineSeries3D 可以在 3D 空间中展示出点与线。对于点来说，有许多基本的 3D 形状可用。如果将 ***LineVisible*** 属性设置为 ***true***，那么点可以用一条线连接起来。

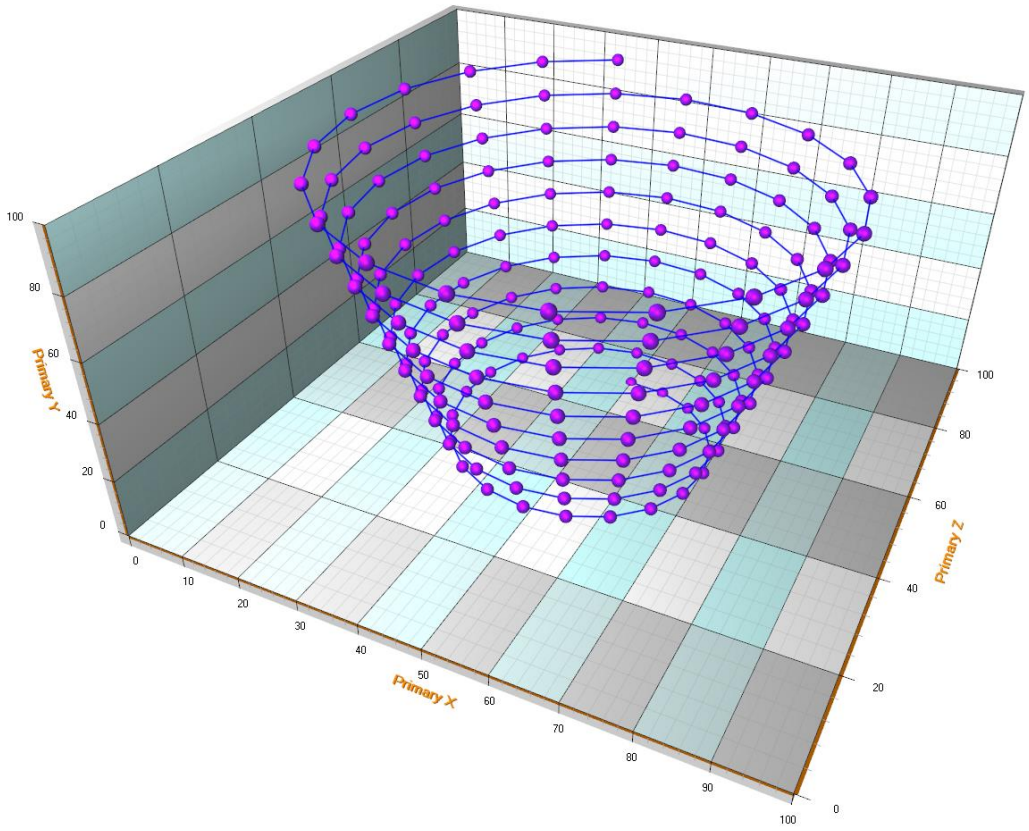


图 7-20. 一个 PointLineSeries3D 示例， PointStyle 的 Shape 设置为 Sphere.

7.9.1 点样式

点可以显示为真实的 3D 点，也可以显示为 2D 形状。

PointStyle	
DetailLevel	30
Rotation3D	
Shape2D	
Angle	**45**
Antialiasing	True
BitmapAlphaLevel	255
BitmapImage	**(none)**
BitmapImageTintColor	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	

图 7-21. PointStyle 属性树。ShapeType 可用以在 2D 与 3D 形状之间进行切换。

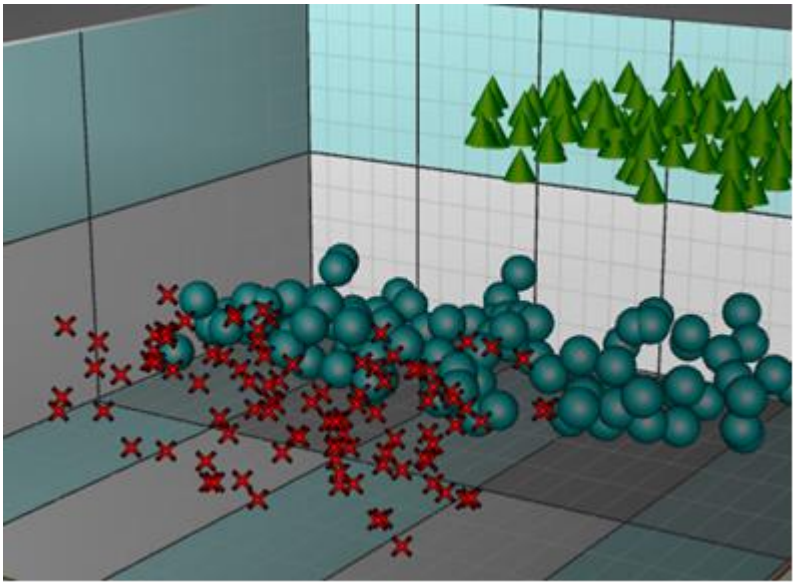


图 7-22. 对红色十字设置 ShapeType = Shape2D. 青色与绿色设置 ShapeType = Shape3D.

注意! 2D 形状是在所有 3D 对象上渲染的，它们不支持基于其他对象可见性来隐藏。

7.9.2 线条样式

LineStyle	
AntiAliasing	**Normal**
Color	Yellow
LineOptimization	**Hairline**
Pattern	Solid
PatternScale	1
Width	**0.2**

图 7-23. LineStyle properties.

线条可渲染为 3D 线条，或者一根 1 像素宽的发线。

当在系列中有许多数据时，建议设置 ***LineOptimization = Hairline*** 来避免出现性能问题。

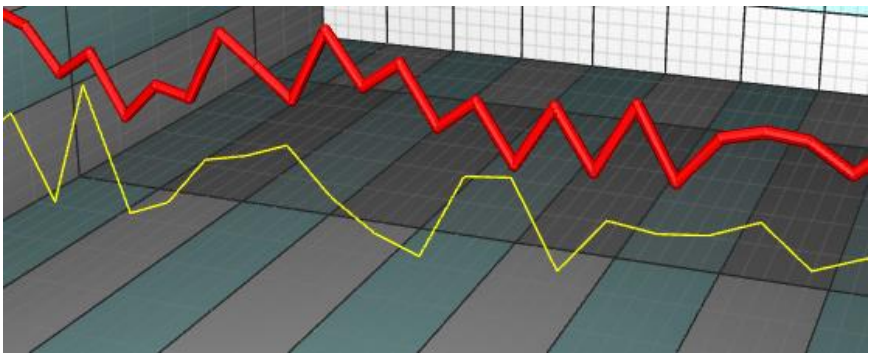


图 7-24. 黄线： LineStyle.LineOptimization = Hairline. 红线： LineStyle.LineOptimization = NormalShading.

除了 ***LineStyle*** 设置外，***PointLineSeries3D*** 还具有 ***ClosedLine*** -属性，当启用该属性时，会自动连接该系列的第一个和最后一个数据点。从 LightningChart 9.0 版开始此属性可用。

```
//连接第一点和最后一点
pointLineSeries3D.ClosedLine = true;
```

优化半透明线，见第 7.10.9 章

7.9.3 添加点

PointLineSeries3D 支持三种不同的点格式：

- Points s 属性 （SeriesPoint3D 数组）
- PointsCompact 属性（SeriesPointCompact3D 数组）
- PointsCompactColored 属性（SeriesPointCompactColored3D 数组）

PointsCompact 和 ***PointsCompactColored*** 结构可非常有效利用内存，可以利用简单的点样式实现多达 1 亿个数据点可视化。通过 **PointsType** 属性可以设置点的格式。

7.9.3.1 点

使用 ***Points*** 属性，可以支持所有高级的着色选项。SeriesPoint3D 结构包括以下字段：

double X: X 轴值
double Y: Y 轴值
double Z: Z 轴值
Color color: 单个数据点颜色，仅当开启 ***IndividualPointColors*** 或 ***MultiColorLine*** 后可应用。
float sizeFactor: 尺寸系数乘以由 ***PointStyle.Size*** 定义的尺寸大小。仅当开启 ***IndividualPointSizes*** 后可应用。
object Tag: 可自由分配的辅助对象，例如附加一些细节。

系列点必须以代码进行添加。用 ***AddPoints（…）*** 方法在现有点的末端添加点。

```
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）; //在末端添加点
```

若要一次性设置全部系列数据，并覆盖旧的点，可以直接分配新点数组:

```
chart.View3D.PointLineSeries[0].Points = pointsArray;  //分配点数组
```

7.9.3.2 PointsCompact

PointsCompact 属性能够使内存消耗低，这在拥有大量数据点时非常重要。

SeriesPointCompact3D 结构包括以下字段：

float X: X 轴值
float Y: Y 轴值
float Z: Z 轴值

```
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）; //在末端添加点
```

若要一次性设置全部系列数据，并覆盖旧的点，可以直接分配新点数组:

```
chart.View3D.PointLineSeries[0].PointsCompact = pointsArray;  //分配点数组
```

7.9.3.3 PointsCompactColored

PointsCompactColored 属性能够使内存消耗低，这在拥有大量数据点时非常重要，但仍然可以用单独的颜色对点进行着色。

SeriesPointCompactColoured3D 结构包含以下字段：

float X: X 轴值
float Y: Y 轴值
float Z: Z 轴值
int Color: 点的颜色

```
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）; //在末端添加数据
```

若要一次性设置全部系列数据，并覆盖旧的点，可以直接分配新点数组:

```
chart.View3D.PointLineSeries[0].PointsCompactColored = pointsArray; //分配点数组
```

7.9.4 单独对点着色

设置 ***IndividualPointColors = True***，点的颜色字段应用，代替了 ***Material.DiffuseColor***

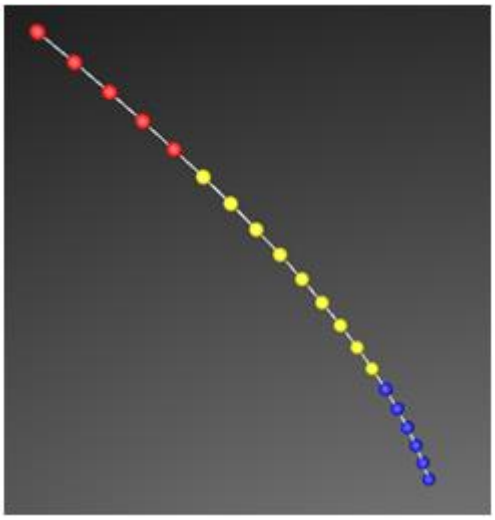


图 7-25. IndividualPointColors 应用示例

注意!当设置 PointsType = PointsCompact 时，不支持单独点着色。

7.9.5 单独设置点大小

设置 ***IndividualPointSizes = True***, 点的 ***sizeFactor*** 字段生效。系数乘以在中 ***PointStyle.Size*** 定义的尺寸大小。

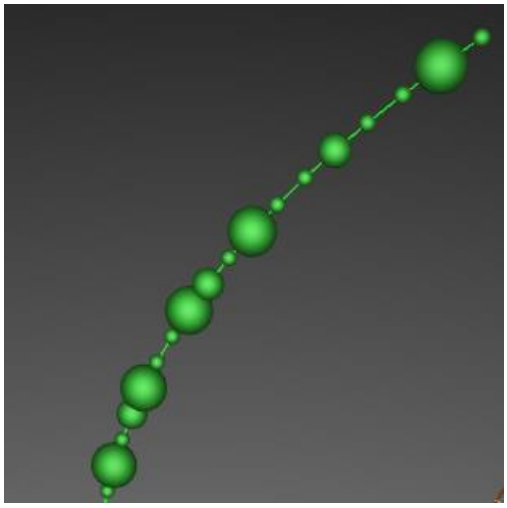


图 7-26. IndividualPointSizes 应用示例

注意! 当设置 PointsType = PointsCompact, 或 PointsCompactColored 时，不支持单独点着色。

7.9.6 多色线

用给定的数据点颜色对线着色，可设置 ***MultiColorLine = True***。图表在相邻点之间内插颜色渐变。

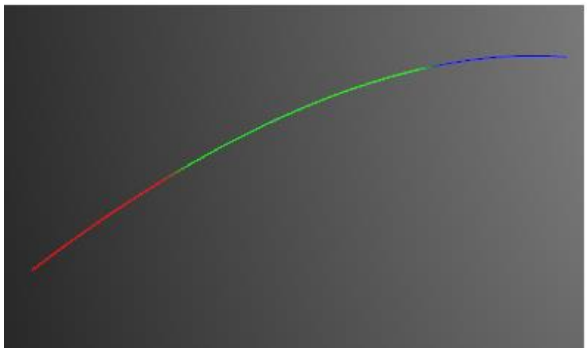


图 7-27. MultiColorLine 开启.

注意! 当设置 PointsType = PointsCompact 时，不支持 MultiColorLine。

7.9.7 显示无数的散点

演示示例： Point cloud

设置 ***PointsOptimization = Pixels***，能够显示数量非常大的散点。每个系列点将会渲染为一个单像素。若要显示 1000 万或 1 亿个数据点，可使用 ***PointsCompact*** （参阅第 7.9.3.2 章节）或 ***PointsCompactColored*** （参阅第 7.9.3.3 章节）方法来保持内存需求发挥作用。

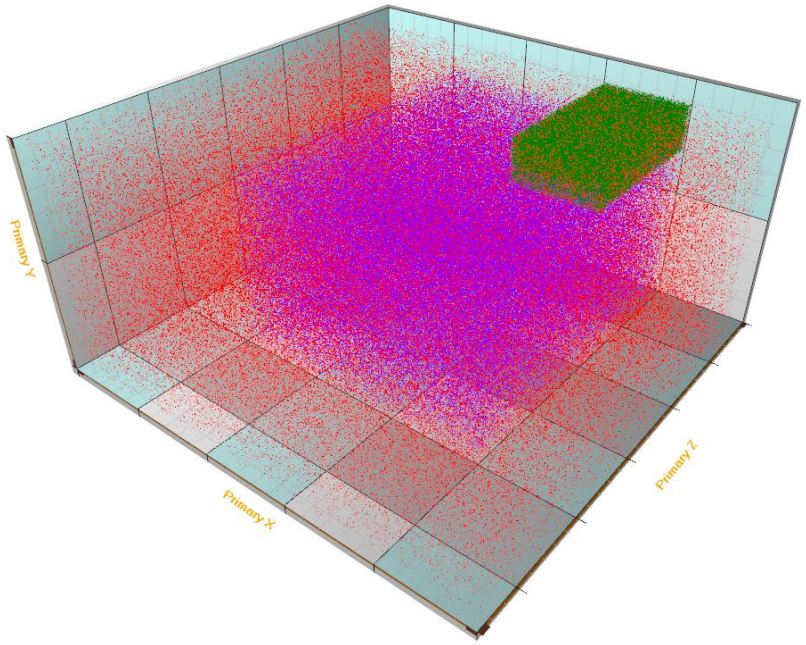


图 7-28. 无数的散点. LineVisible = False, PointsVisible = True, PointsOptimization = Pixels.

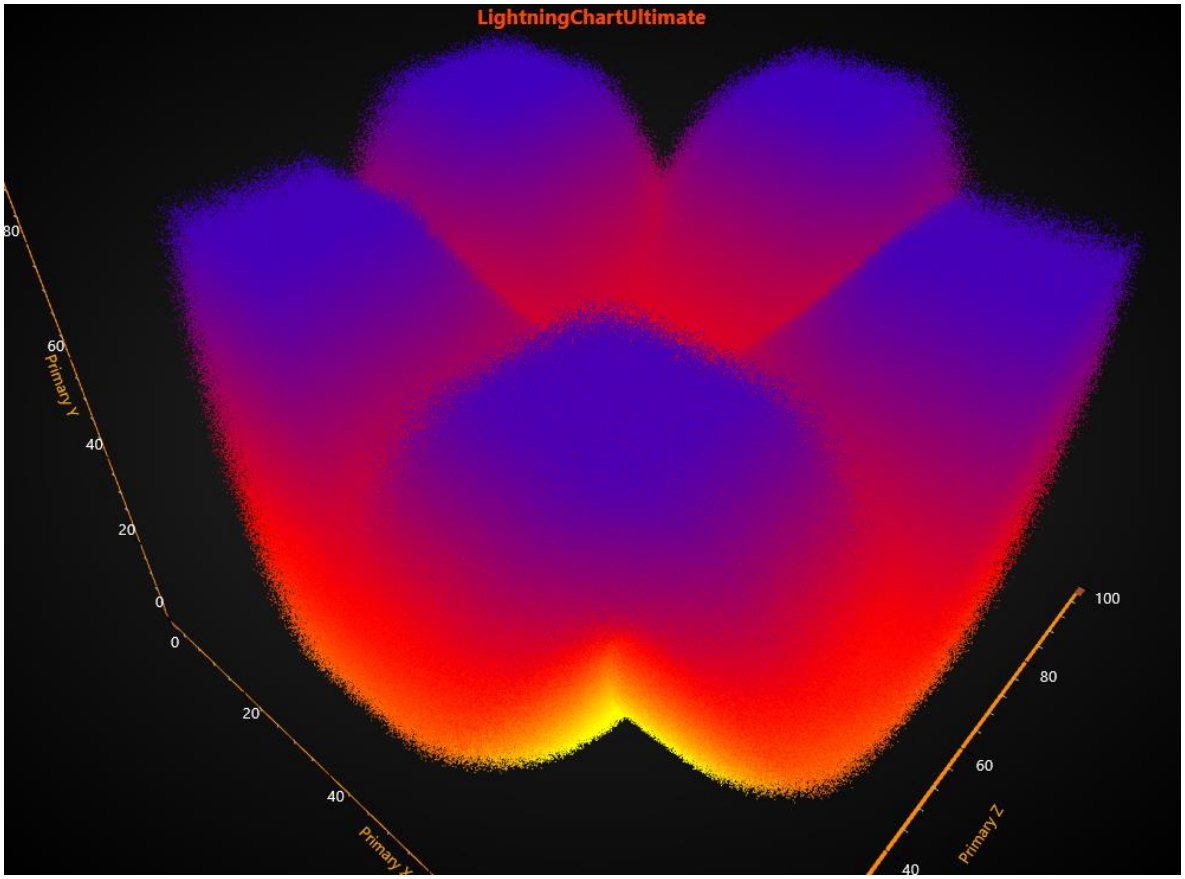


图 7-29. IndividualPointsColoring = True, 使用 PointsCompactColored, LineVisible = False, PointsVisible = True；1.2 亿个分散点

用矩形可以最有效地显示数百万个数据点。当使用 ***PointsCompactColored*** 或 ***PointsCompact***, 点的大小可以用来 ***PointStyle.Shape2D.Width*** 和 ***PointStyle.Shape2D.Height*** 控制。

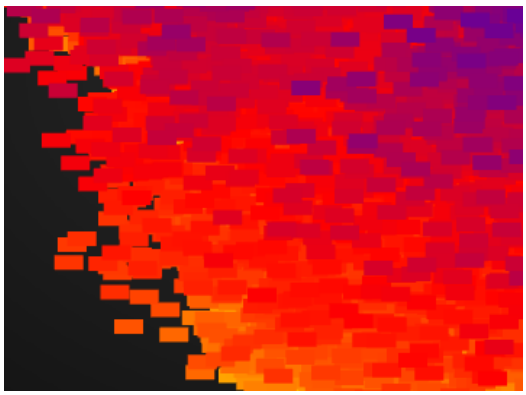


图 7-30. PointStyle.Shape2D， Width = 20 且 PointStyle.Shape2D.Height = 10.

7.10 SurfaceGridSeries3D

演示示例： Simple 3D surface grid; Surface grid, flat projections; Surface grid, value coloring; Surface grids, water and ground

SurfaceGridSeries3D 可以将数据可视化为一个 3D 面。在 ***SurfaceGridSeries3D*** 中, 节点在 X 维度上是等距的，在 Z 维度上也是如此。

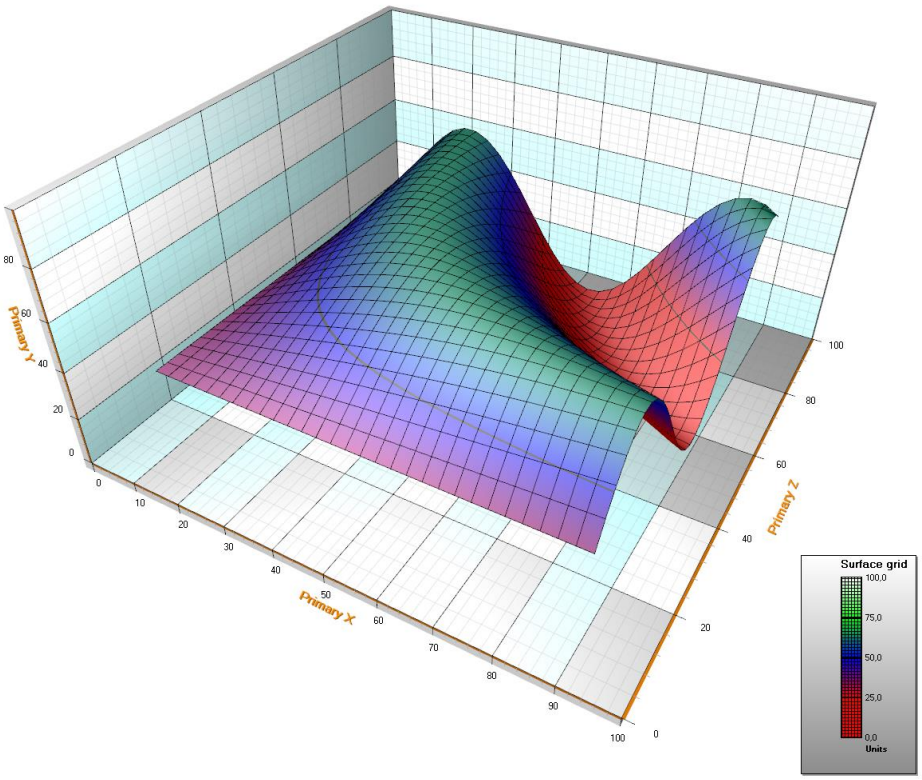


图 7-31. 采用默认样式的面网格系列。高度数据用正弦公式表示。图例框显示高度着色间隔。

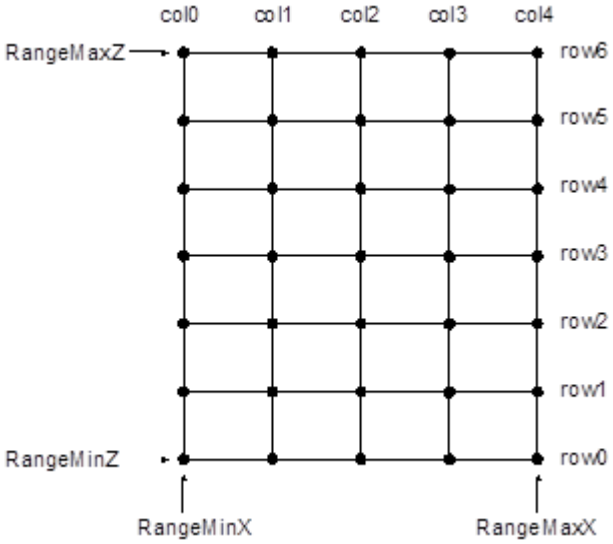


图 7-32. 面网格节点。SizeX = 5, SizeZ = 7.

节点距离自动计算如下：

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

$$node\ distance\ Z = \frac{\text{RangeMaxZ} - \text{RangeMinZ}}{\text{SizeZ} - 1}$$

7.10.1 设置面网格数据

- 用 ***RangeMinX*** 和 ***RangeMaxX*** 属性来设置 X 轴值域，根据指定的 X 轴对最小值和最大值排序。
- 用 ***RangeMinZ*** 和 ***RangeMaxZ*** 属性来设置 Z 轴值域，根据指定的 Z 轴对最小值和最大值排序。
- 设置 ***SizeX*** 和 ***SizeZ*** 属性，将网格的尺寸大小设置为以列和行表示。
- 为所有节点设置 Y 值：

采用数据数组索引的方法

```
for （int nodeIndexX = 0; nodeIndexX < columnCount; nodeIndexX ++）
{
        for （int nodeIndexZ = 0; nodeIndexZ < rowCount; nodeIndexZ ++）
        {
          Y =//一些高度值
          gridSeries.Data[iNodeX, iNodeZ].Y = Y;
        }
}
gridSeries.InvalidateData（）; // 准备好新值后通知刷新
```

采用 SetDataValue 的另一种方法

```
for （int nodeIndexX = 0; nodeIndexX < columnCount; nodeIndexX ++）
{
        for （int nodeIndexZ = 0; nodeIndexZ < rowCount; nodeIndexZ ++）
        {
          Y =//一些高度值
          gridSeries.SetDataValue（nodeIndexX, nodeIndexX,
                    0, // X值与网格无关
                    Y,
                    0,// Z值与网格无关
                    Color.Green）; //本例中没有使用源点颜色，所以这里用什么颜色都行
        }
}
gridSeries.InvalidateData（）; // 准备好新值后通知刷新
```

7.10.2 根据位图文件创建曲面

演示示例： ***Large surface***

用 ***SetHeightDataFromBitmap*** 方法可根据位图图像来创建曲面。面可获得位图尺寸（若没有使用反锯齿或重新采样）。对于每个位图图像像素，对红色、绿色和蓝色值求和。和越大，该节点的高度数据值就越高。黑色与暗色的值较低，而明亮和白色的值较高。

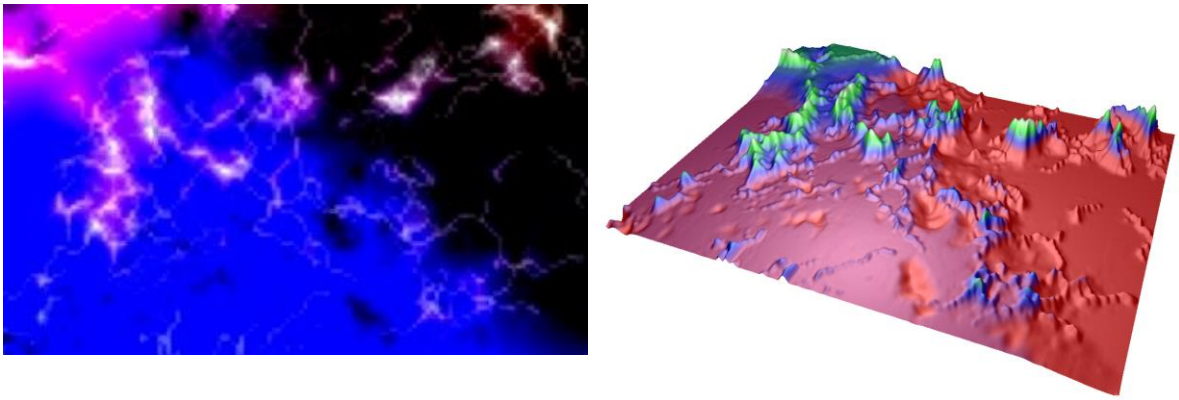


图 7-33. 源位图与计算所得的曲面高度数据。暗色值较低，而亮色值较高。

7.10.3 填充样式

用 ***Fill*** 属性来选择曲面的填充样式。可以使用以下选项：

- ***None***: 使用此选项，不进行填充。这一选项对线框网眼很有用。
- ***FromSurfacePoints***: 使用数据属性节点的颜色。
- ***Toned***: 应用 ToneColor
- ***PalettedByY***: 根据调色板按 Y 值着色，参阅第 **Error! Reference source not found.**章节
- ***PalettedByValue***: 根据调色板按 **SurfacePoint's Value** 字段着色，参阅第 **Error! Reference source not found.**章节
- ***Bitmap***: 将位图图像拉伸来覆盖整个曲面。在 ***BitmapFill*** 属性中对位图图像进行设置。属性具有子属性可用于进行垂直和水平镜像。

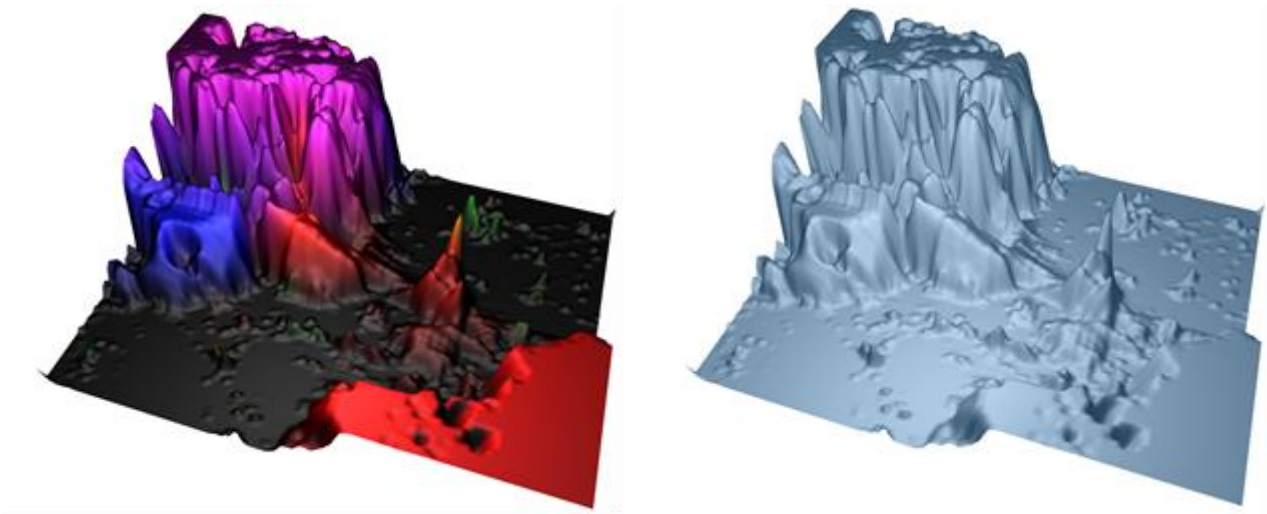


图 7-34. FromSurfacePoints 填充；每个数据点的颜色。

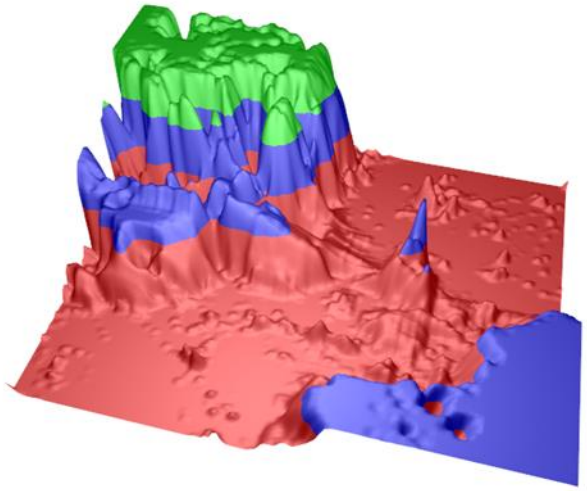


图 7-35. Toned 填充.

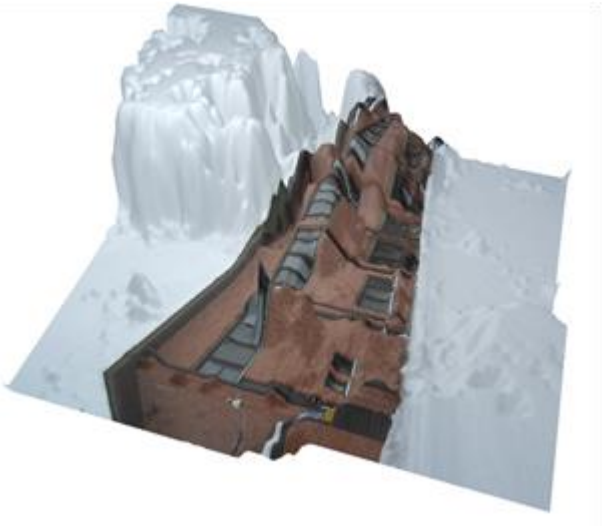


图 7-36. PalettedByY

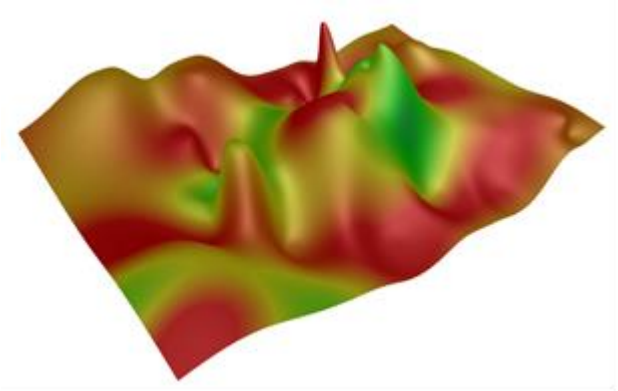


图 7-37. Bitmap 填充.

图 7-38. PalettedByValue.

7.10.4 轮廓调色板

ContourPalette 属性可以为高度着色定义颜色阶。***ContourPalette*** 可用于:

- ***Fill*** （参阅第 **Error! Reference source not found.** 章节）
- ***Wireframe mesh*** （参阅第 **Error! Reference source not found.** 章节）
- ***Contour lines*** （参阅第 **Error! Reference source not found.** 章节）

为轮廓调色板可以定义无限数量的色阶。每一阶都有一个高度值以及一个对应的颜色。

调色板包括 ***MinValue***、***Type*** 和 ***Steps*** 属性。***Type*** 属性包括两个选择： ***Uniform*** 和 ***Gradient***。图 7-39 的轮廓调色板显示了如下信息：

- ***MinValue***: 0
- ***Type***: Gradient （渐变）
- ***Steps***:
  - [0]阶: MaxValue（最大值）: 25, Color（颜色）: Red 红色

- [1] 阶: MaxValue（最大值）: 50, Color（颜色）: Blue 蓝色
- [2] 阶: MaxValue（最大值）: 75, Color（颜色）: Lime 绿黄色
- [3] 阶: MaxValue（最大值）: 100, Color（颜色）: White 白色

使用第一阶的颜色给第一阶值下面的高度值着色。

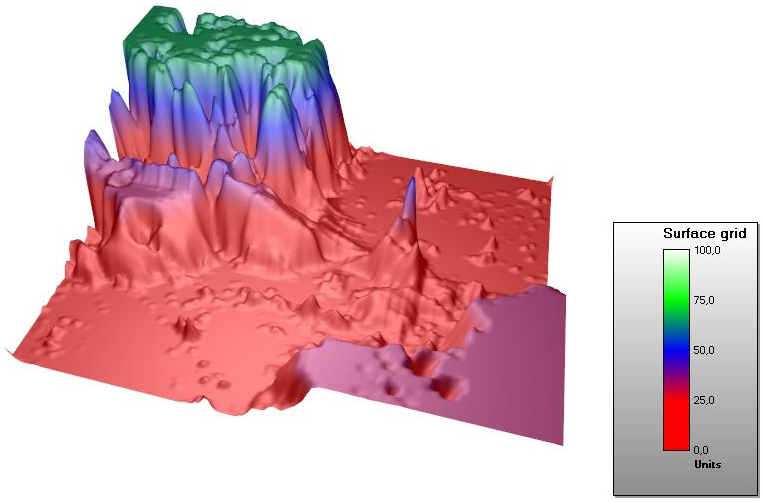


图 7-39. 曲面网格系列轮廓调色板样式（Type）设置为渐变（Gradient）。

7.10.5 线框网眼

用 ***WireframeType*** 来选择线框样式。选项有：

- ***None***: 无线框
- ***Wireframe***: 一个纯色线框。可以用 ***WireframeLineType.Color*** 来设置颜色。
- ***WireframePalettedByY***: 线框着色依据 SurfacePoint 的 ***Y*** 字段 ***ContourPalette*** 进行（参阅第 **Error! Reference source not found.**章节）
- ***WireframePalettedByY***: 线框着色依据 SurfacePoint 的 ***Value*** 字段 ***ContourPalette*** 进行（参阅第 **Error! Reference source not found.**章节）
- ***WireframeSourcePointColored***: 线框着色依据曲面节点的颜色进行
- ***Dots***: 在节点位置绘制纯色点。
- ***DotsPalettedByY***: 在节点位置绘制点，并根据 SurfacePoints 的 ***Y*** 字段按照 ***ContourPalette*** 来着色。
- ***DotsPalettedByValue***: 在节点位置绘制点，并根据 SurfacePoints 的 ***Value*** 字段按照 ***ContourPalette*** 来着色。
- ***DotsSourcePointColored***: 在节点位置绘制点，依据曲面节点颜色来着色

线框线条样式（***color***、***width***、***pattern***）可通过 ***WireframeLineStyle*** 来编辑。

注意! 调色板彩色线框线和点可能与 WireframeLineStyle.Patternsettings 冲突，例如线框设置为 Dots 的 Dash 线条样式。 请在其中一个中使用实线。

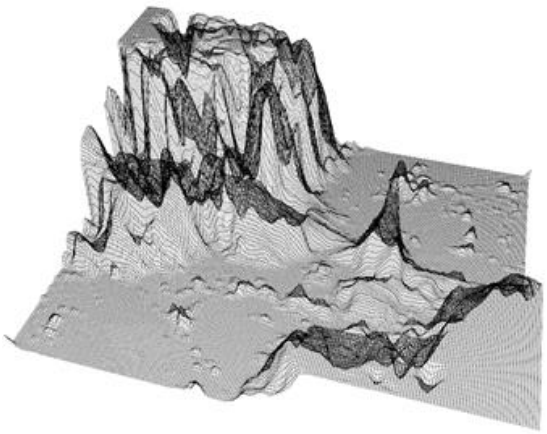


图 7-40. WireframeType = Wireframe.

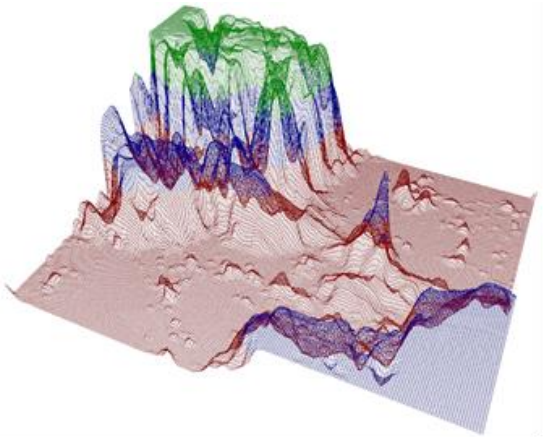


图 7-41. WireframeType = WireframePalettedByY.

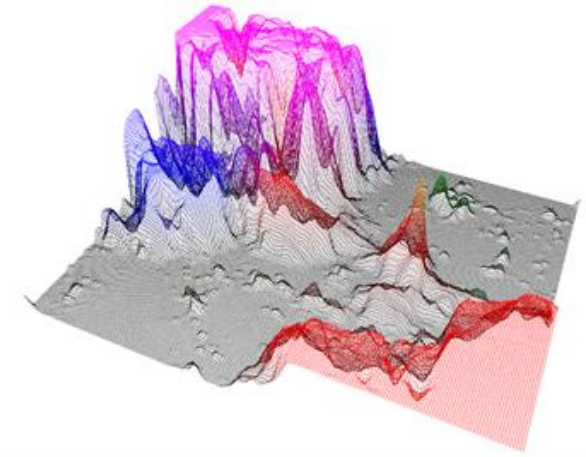


图 7-42. WireframeType = SourcePointColored.

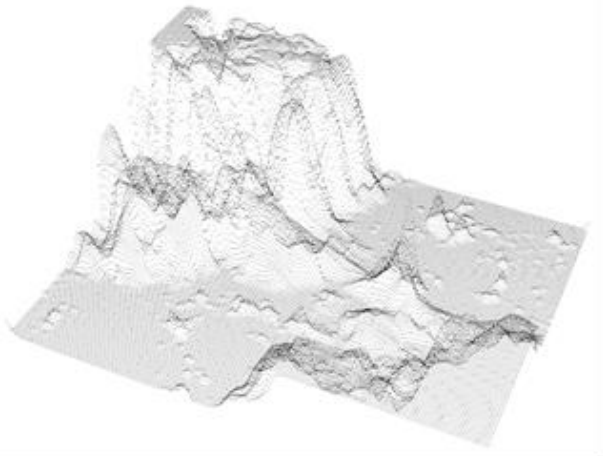


图 7-43. WireframeType = Dots.

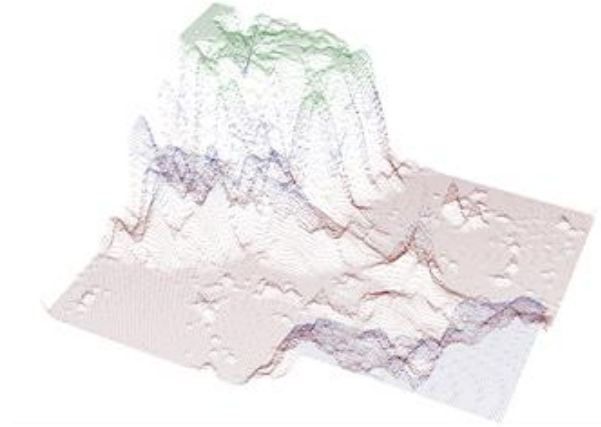


图 7-44. WireframeType = DotsPalettedByY.

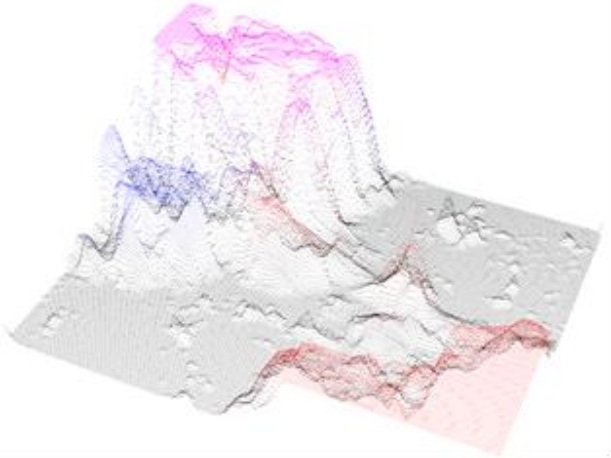


图 7-45. WireframeType = DotsSourcePointColored.

7.10.5.1 线框与填充同时使用的注意事项

当在 3D 模型中同样的位置绘制填充与线框时，可能会出现深度冲突（Z-Fighting）现象。这可以看成是线框线条中断。这是因为 GPU 不可能判断出哪个对象更靠近摄像头。

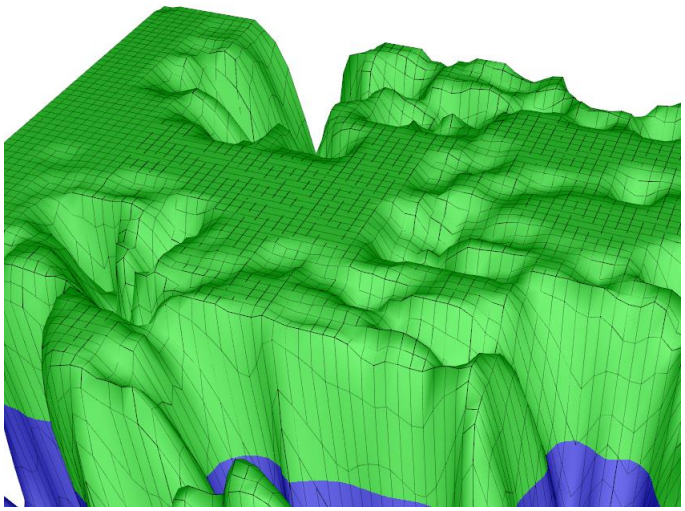


图 7-46. 采用填充的曲面网格线框。Z-fighting 看上去像断损的线框线条。

为避免出现 Z-fighting 现象，可使用 ***WireframeOffset*** 或 ***DrawWireframeThrough*** 属性。利用 ***WireframeOffset,***可在 3D 模型空间中轻微移动线框。 ***DrawWireframeThrough*** 通过填充来绘制线框，无论该表面的部分是否对摄像头可见。

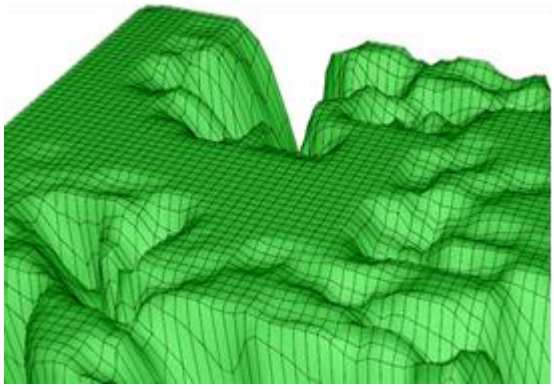


图 7-47. WireframeOffset = (X=0; Y=0.1; Z=0).

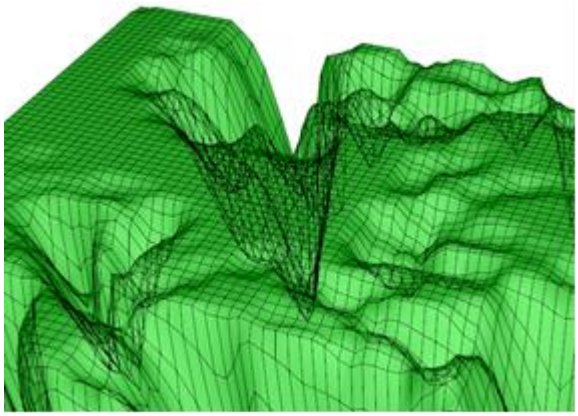


图 7-48. DrawWireframeThrough 开启.

7.10.6 轮廓线

轮廓线可以快速表明高度数据，而不用采取调色板填充来填充曲面。轮廓线可以与填充线框结合使用。通过设置 ***ContourLineType*** 属性，可以用不同的样式来绘制轮廓线：

- ***None***: 不显示轮廓线
- ***FastColorZones***: 轮廓线可绘制为细的直立带。这可以进行非常强的渲染，很适合不断更新或动画表面。陡峭的高度变化以细线表示，平缓倾斜的高度差以粗线表示。所有线条都使用由 ***ContourLineStyle.Color*** 属性定义的同样颜色。带的高度可以用 ***FastContourZoneRange*** 属性来设置。

- ***FastPalettedZones***: 和 ***FastColorZones*** 类似，但是线条着色是依据 ***ContourPalette*** 选项进行（参阅第 7.10.4 章节）。
- ***ColorLineByY*** 和 ***ColorLineByValue***: 轮廓线是由实际线制成的。渲染用时比 ***FastColorZones*** 久。 线条宽度可以用 ***ContourLineStyle.Width*** 属性来调节。轮廓线也能用 ***WireframeOffset*** 属性进行移位，以消除填充时可能出现的 Z-fighting 现象。
- ***PalettedLineByY*** 和 ***PalettedLineByValue***: 与 ***ColorLineByY*** 和 ***ColorLineByValue*** 类似，但是线条着色依据 ***ContourPalette*** 选项进行（参阅第 7.10.4 章节）。

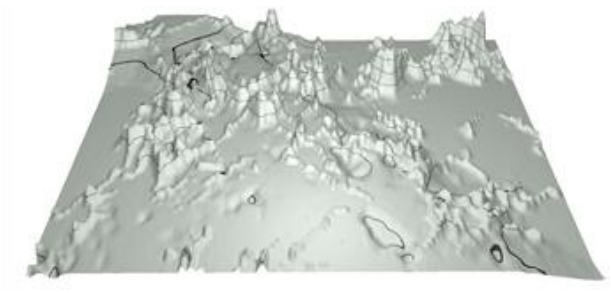


图 7-49. ContourLineType = FastColorZones.

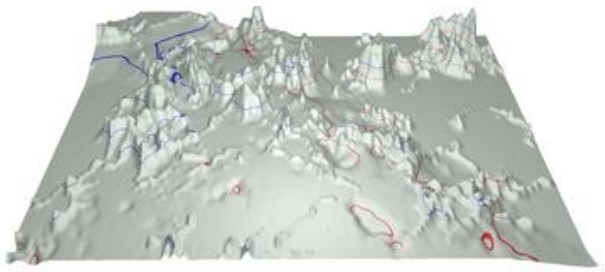


图 7-50. ContourLineType = FastPalettedZones.

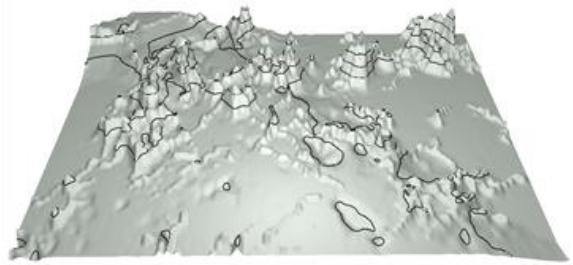


图 7-51. ContourLineType = ColorLine.

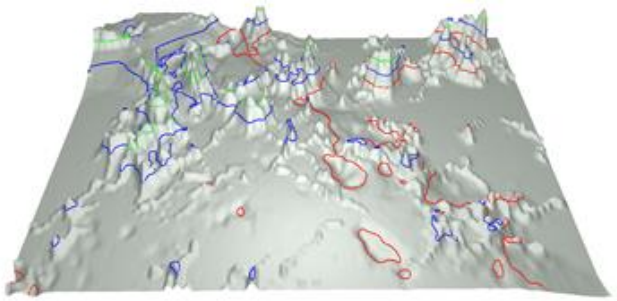


图 7-52. ContourLineType = PalettedLine.

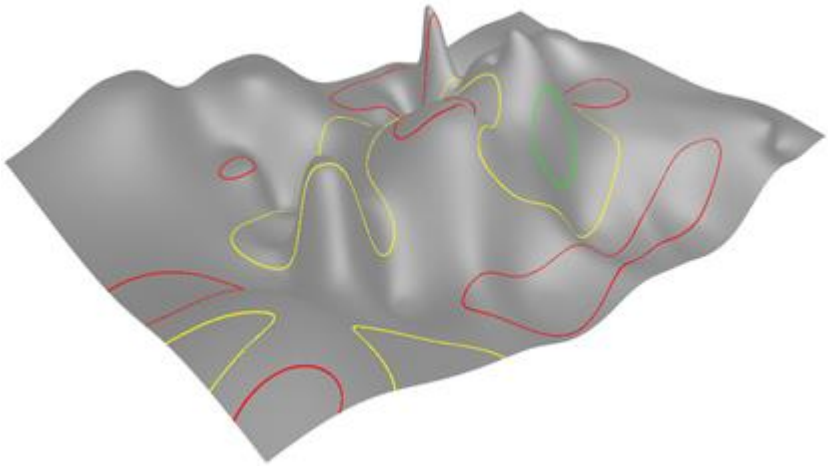


图 7-53. ContourLineType = PalettedLineByValue.

7.10.7 消退

演示示例： Spectrum 3D

SurfaceGridSeries3D、***SurfaceMeshSeries3D*** 和 ***WaterfallSeries3D*** 具有一种 ***FadeAway*** 属性，可以将系列向图表的后部淡出。 ***Fadeaway*** 是以百分比计算，有效范围从 0 （默认值，无消退）到 100（完全消退）。值越高，具有高 Z 值的数据越透明。

7.10.8 滚动曲面数据

演示示例： Spectrogram

SurfaceGridSeries3D 和 ***SurfaceMeshSeries3D*** 具有 ***InsertRowBackAndScroll*** 和 ***InsertColumnBackAndScroll*** 方法，可用于性能优化的定期数据添加。他们在曲面系列的数据表中插入一个新的数据行或列，同时丢掉最早的值，即除去第一个数据行。思考一下下面的 3D 光谱显示（图 6-54）。新的 FFT 值作为一个新行添加（靠近摄像头），并滚动旧数据与时间轴（Z 轴）。最早的曲面值必须丢弃掉。

InsertRowBackAndScroll 和 ***InsertColumnBackAndScroll*** 将新数据作为双数组，但也需要为曲面系列和滚动轴设置新的最小值和最大值（Z 维度/轴用 ***InsertRowBackAndScroll***，X 维度/轴用 ***InsertColumnBackAndScroll***）. 这种不断调整的系列和轴的范围，使滚动发挥作用。

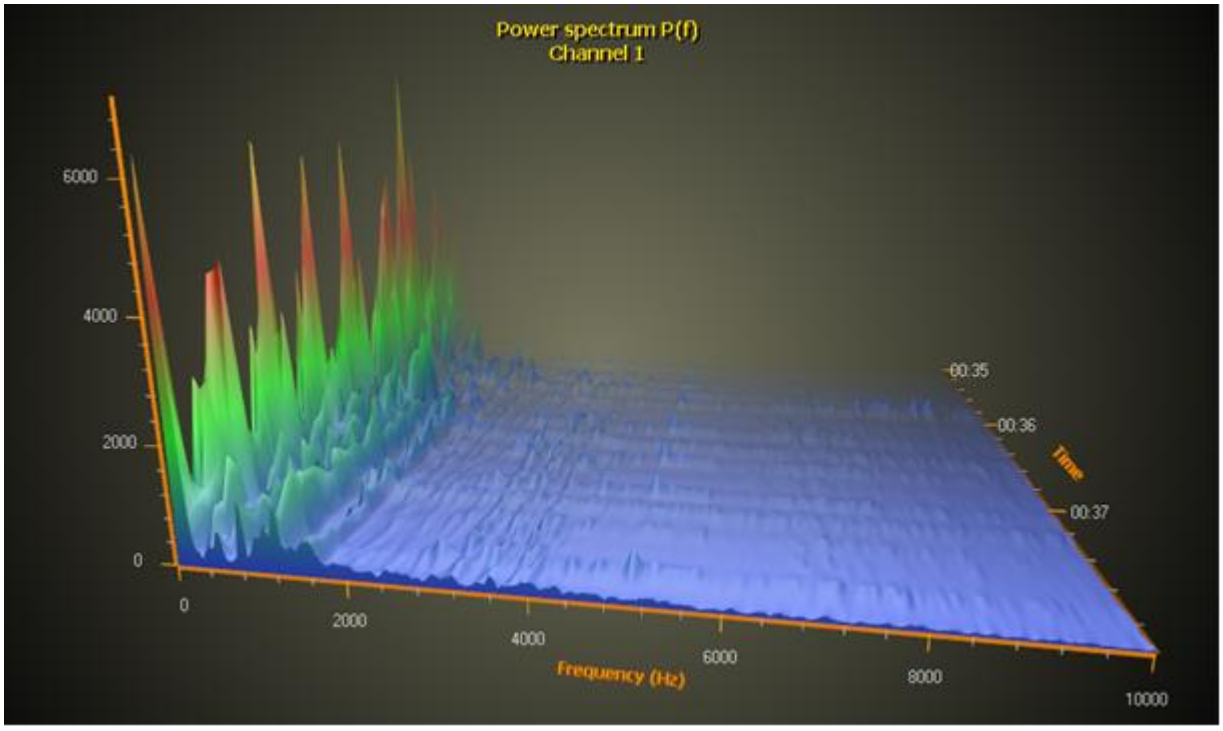


图 7-54. 用曲面网格表示 3D 光谱。InsertRowBackAndScroll 方法用于性能优化的数据添加。Fadeaway 属性设为 100，使表面平滑地向图表的后部消退。采用的是透视摄像头。

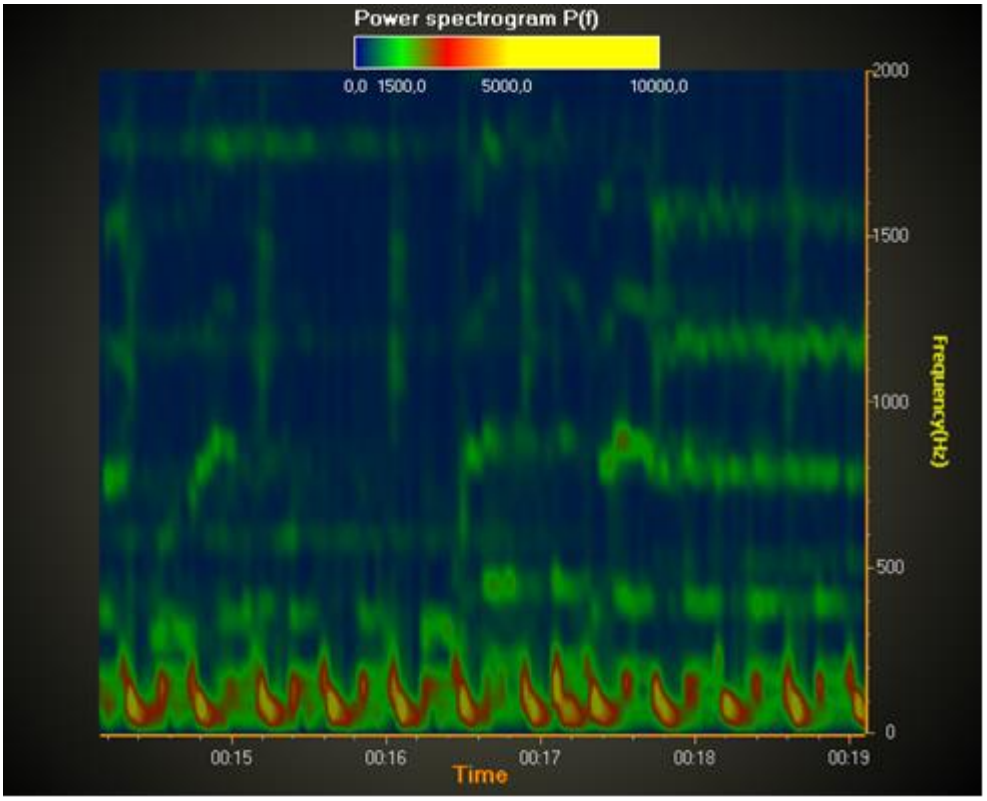


图 7-55. 采用表面网格的光谱图。采用的是 InsertColumnBackAndScroll 方法。在模型上方的正投摄像头给出了连续垂直的投影。 SuppressLighting 开启后可移除不需要的光反射。设置 Fadeaway = 0 可让网格系列完全可见。

7.10.9 透明度处理

虽然渲染不透明表面很简单，但半透明或透明表面的情况会变得稍微复杂一些，因为它们应该允许看到其他 3D 系列和对象以及它们后面的数据点。LightningChart 提供了 3 种处理表面透明度的选项：无序、ShaderApproximation 和 OrderingTriangles，每种都有其优点和缺点。TransparencyRenderMode 属性可用于选择此选项。

无序 - 按创建顺序渲染透明对象面。这适用于所有非透明表面，并且与旧库行为相同。对于半透明表面，它可能在某些视角下工作，但从其他角度看可能完全不透明，或者表面上的灯光效果可能看起来不正确（可能会看到伪影）。

ShaderApproximation - 使用着色器实现透明效果。这种方法几乎与无序一样快，但多个表面之间或同一表面上的部分透明度可以正确处理。缺点是与 Unordered 或 OrderingTriangles 选项相比，表面边缘不太平滑（更粗糙/锯齿）。当在 3D 图表中呈现不同类型的半透明系列/对象时，应使用 TransparencyRenderMode 的 ShaderApproximation 选项，因为它有助于减轻伪影并改善图表中的半透明对象处理。

OrderingTriangles - 按适当的 z 顺序对对象面三角形进行排序。对于具有大量面（100 万或更多）的项目，这会很慢。它也不适用于多个表面（视角应与多个表面顺序匹配）。

从 LightningChart 版本 12.0 开始，TransparencyRenderMode 适用于所有 3D 系列类型。在以前的版本中，只有表面类型系列（SurfaceGridSeries3D、SurfaceMeshSeries3D 和 WaterfallSeries3D）以及 PointLineSeries3D 才有它。请注意，此属性仅在使用 DirectX 11 渲染器时可用，换句话说，当 RendererDeviceType 为 HardwareOnlyD11 或 SoftwareOnlyD11 时可用。

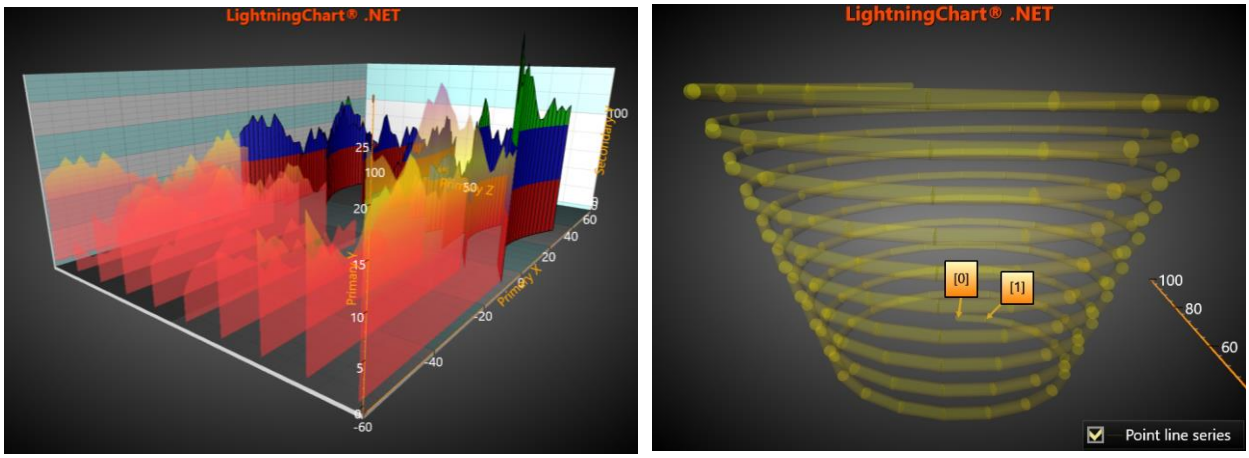


图 7 56. 左侧是两个 WaterfallSeries3D，其中前面的一个使用 ShaderApproximation 进行透明填充。右侧是半透明的 PointLineSeries3D，呈弹簧状盘绕。

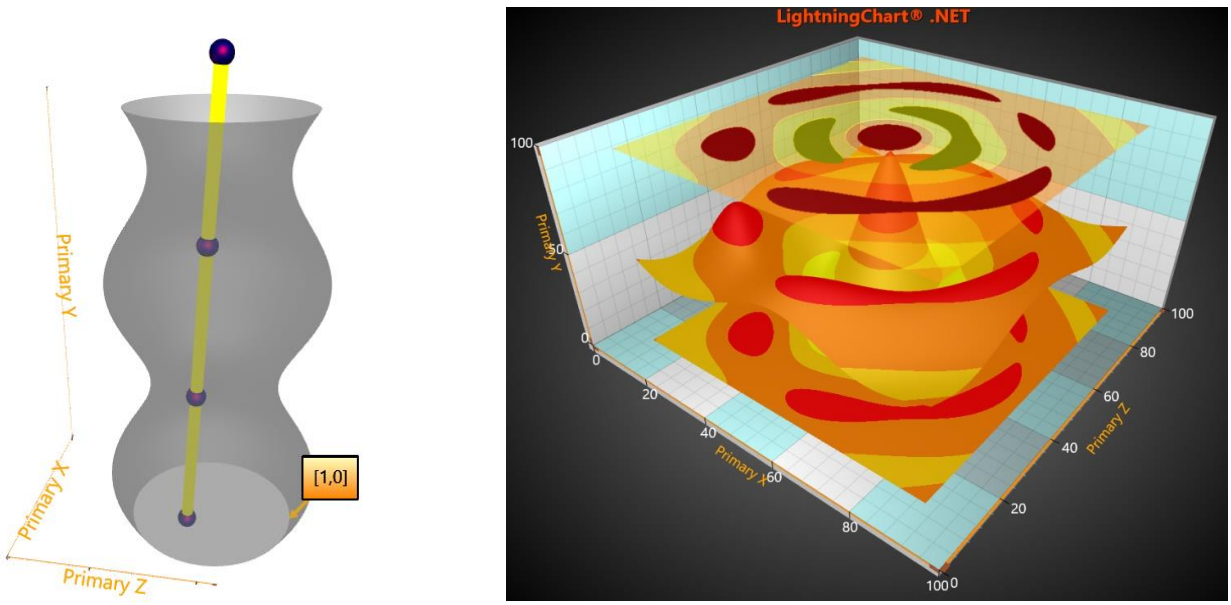


图 7 57. 左侧半透明的 SurfaceMeshSeries3D 被包裹成一个管状。右侧是几个 SurfaceGridSeries3D，其中顶部一个使用 ShaderApproximation 进行透明填充。

7.11 SurfaceMeshSeries3D

演示示例： Surface mesh, heat dissipation; Surface mesh; Stepping surface mesh; Globe with flight routes; Gradient bars

SurfaceMeshSeries3D 与 ***SurfaceGridSeries3D*** 因为主要具有相同的属性，所以二者几乎是相同的。最大的不同之处在于曲面节点可以在 3D 空间中自由放置。换言之，曲面不一定是矩形的。用 ***SurfaceMeshSeries3D*** 可以几乎将曲面弯曲成任何形状，例如一个球形或者一颗人头的形状。

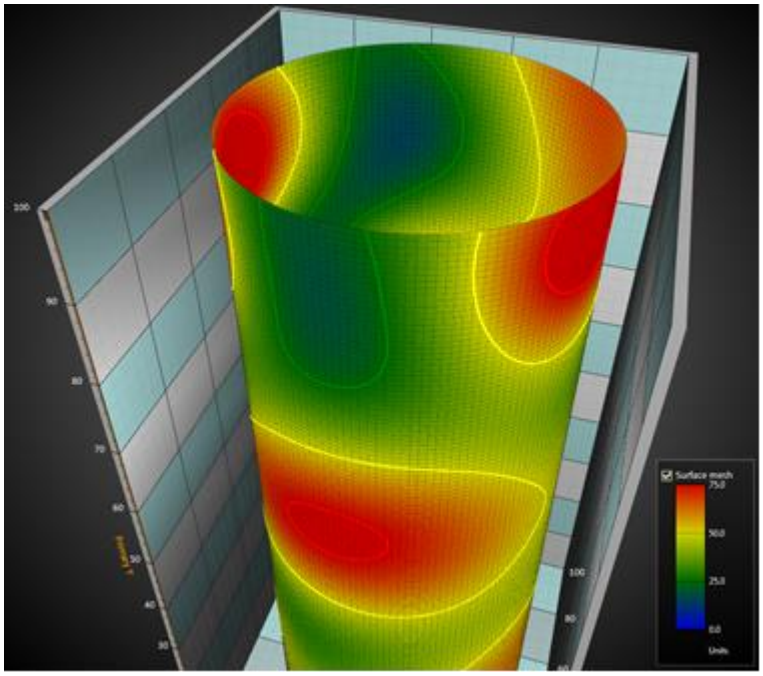


图 7-56. SurfaceMeshSeries3D, 管状几何结构

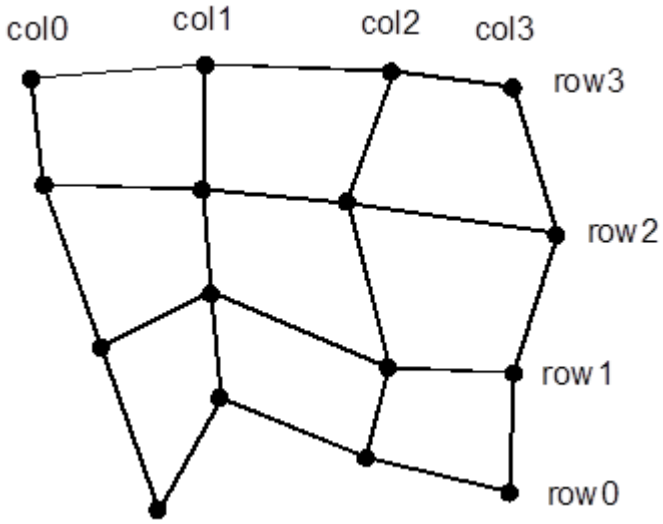


图 7-57. 曲面网眼节点. SizeX = 4, SizeZ =4.

7.11.1 设置曲面网眼数据

- 设置 ***SizeX*** 和 ***SizeZ*** 属性，将网格的尺寸大小设置为以列和行表示。
- 为所有节点设置 X、Y 和 Z 值：

采用数据数组索引的方法

```
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（）; //通知已准备好新值进行刷新
```

采用 SetDataValue 的另一种方法

```
for （int nodeIndexX = 0; nodeIndexX < columnCount; nodeIndexX ++）
{
        for （int nodeIndexZ = 0; nodeIndexZ < rowCount; nodeIndexZ ++）
        {
          meshSeries.SetDataValue（nodeIndexX, nodeIndexZ,
                    xValue,
                    yValue,
                    zValue,
                    dataValue,
                    Color.Green）; //本例中没有使用源点颜色，所以这里用什么颜色都行
        }
}
meshSeries.InvalidateData（）; //通知已准备好新值进行刷新
```

7.12 WaterfallSeries3D （瀑布式系列 3D 视图）

演示示例： Waterfall 3D

采用 ***WaterfallSeries3D***，数据以带形区域实现可视化。还可以像 ***SurfaceGridSeries3D*** 一样对该区域填充，绘制线框与轮廓线，可参阅第 7.10 章节。在 Y 维度中，这种区域从 ***BaseLevel*** 属性值开始。节点数据可以像在 ***SurfaceMeshSeries3D*** 中一样设置，参阅第 **Error! Reference source not found.** 章节。

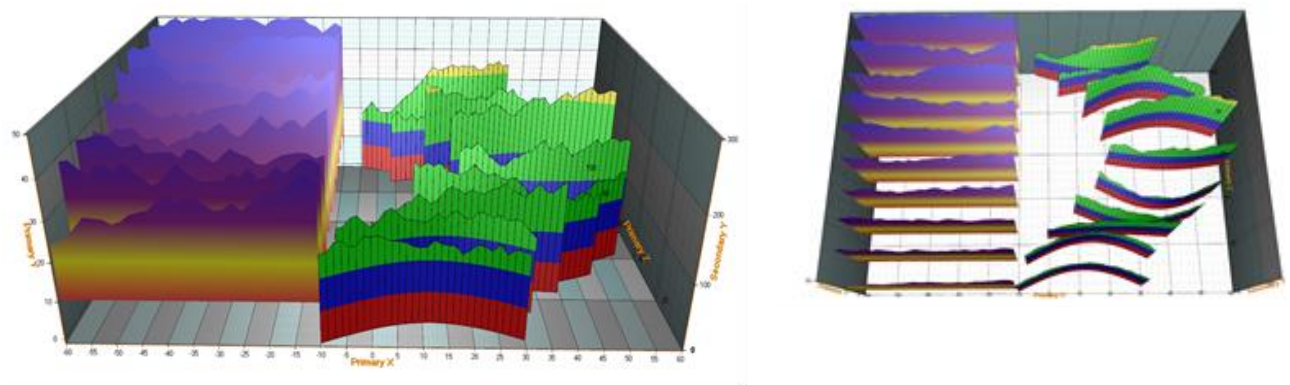


图 7-58. 两个瀑布式（waterfall）系列。在左侧紫罗兰色系列，X 与 Z 呈直角坐标型式；BaseLevel = 10. 右侧红绿蓝系列，X 和 Z 值呈弯曲变化，每一行被放置在不同的水平位置。

在展现传统的 3D 频谱图时，***WaterfallSeries3D*** 特别方便。

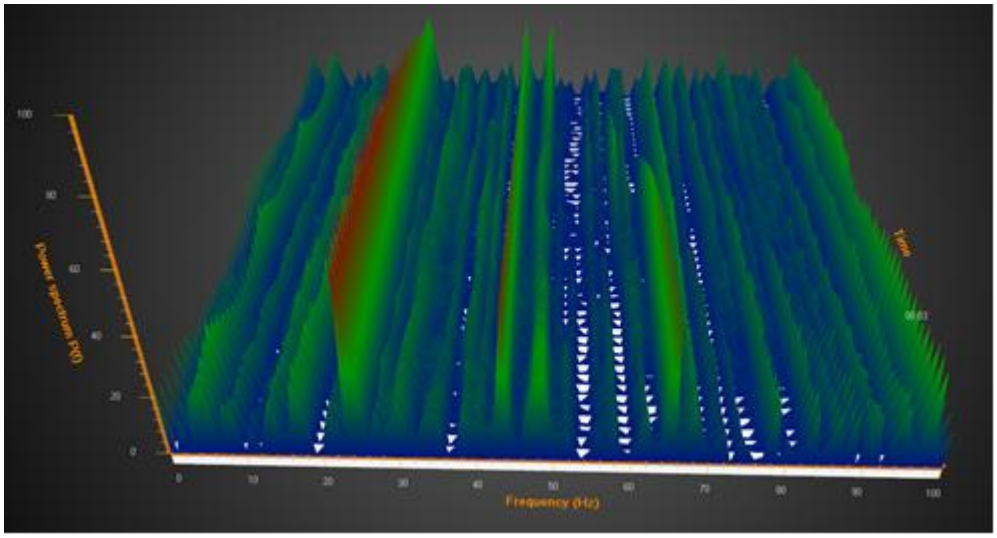


图 7-59. 用 Waterfall 系列展现传统的频谱图

7.13 BarSeries3D

演示示例： Horizontal bars; Bars, grouping; Bars, manhattan

BarSeries3D 可以以 3D 视图实现柱状形数据可视化。

7.13.1 柱状分组

在 View3D 的 ***BarViewOptions*** 属性中，可以利用许多不同选项对柱状系列进行分组。用 ***BarViewOptions.ViewGrouping*** 可控制如何在 3D 视图中对柱状进行分组。

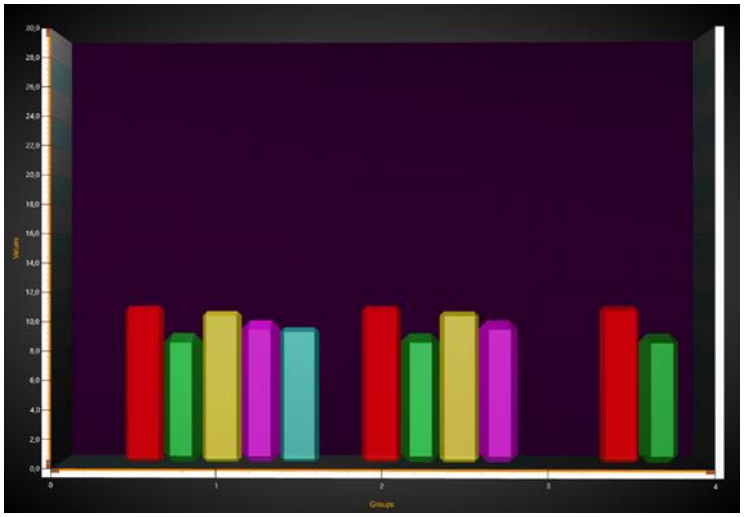


图 7-60. BarViewOptions.ViewGrouping = GroupedIndexedFitWidth.根据柱状索引对其进行分组。布置柱状宽度与各组间距，以更好的适应宽度。

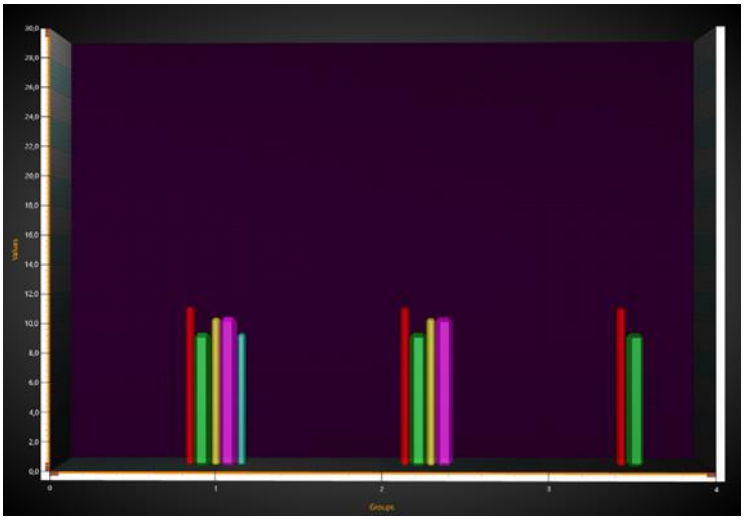


图 7-61. BarViewOptions.ViewGrouping = GroupedIndexed. 应用原柱状宽度，布置各组位置以适应图表宽度。

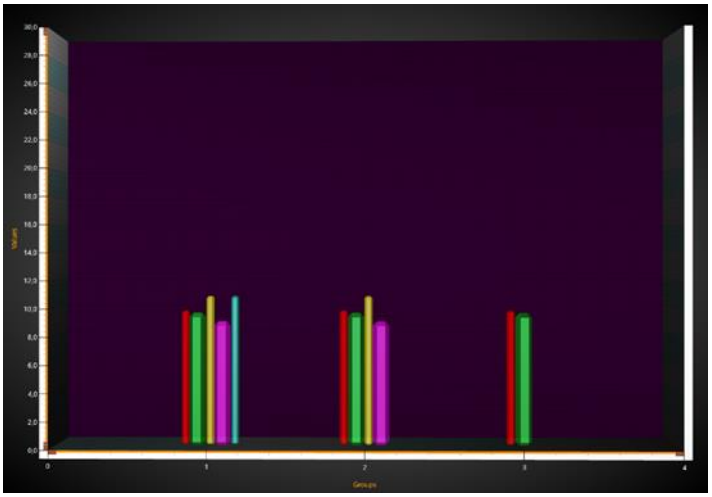


图 7-62. BarViewOptions.ViewGrouping = GroupedByXValue. 柱状 X 值适用.

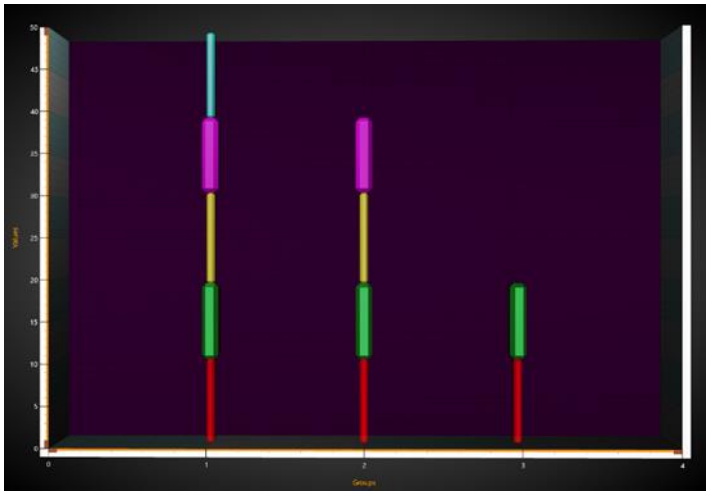


图 7-63. BarViewOptions.ViewGrouping = StackedIndexed. 将所有具有相同索引的柱状进行堆叠。

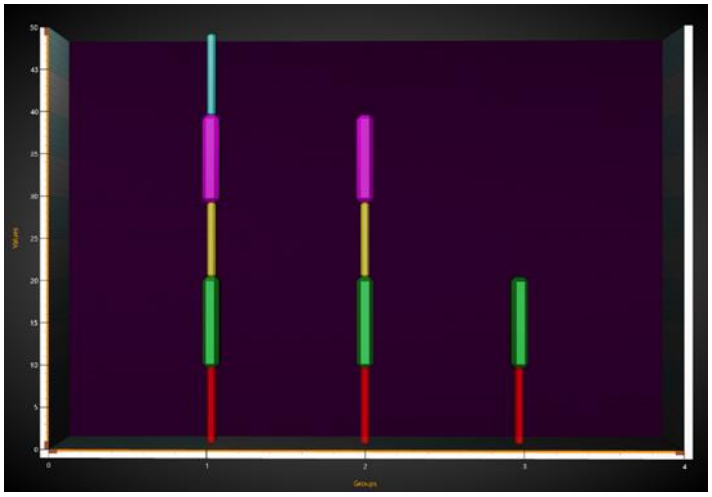


图 7-64. BarViewOptions.ViewGrouping = StackedByXValue. 将所有具有相同 X 值的柱状进行堆叠。此示例与 StackedIndexed 的示例看起来是一样的，因为 X 值和索引是相同的。

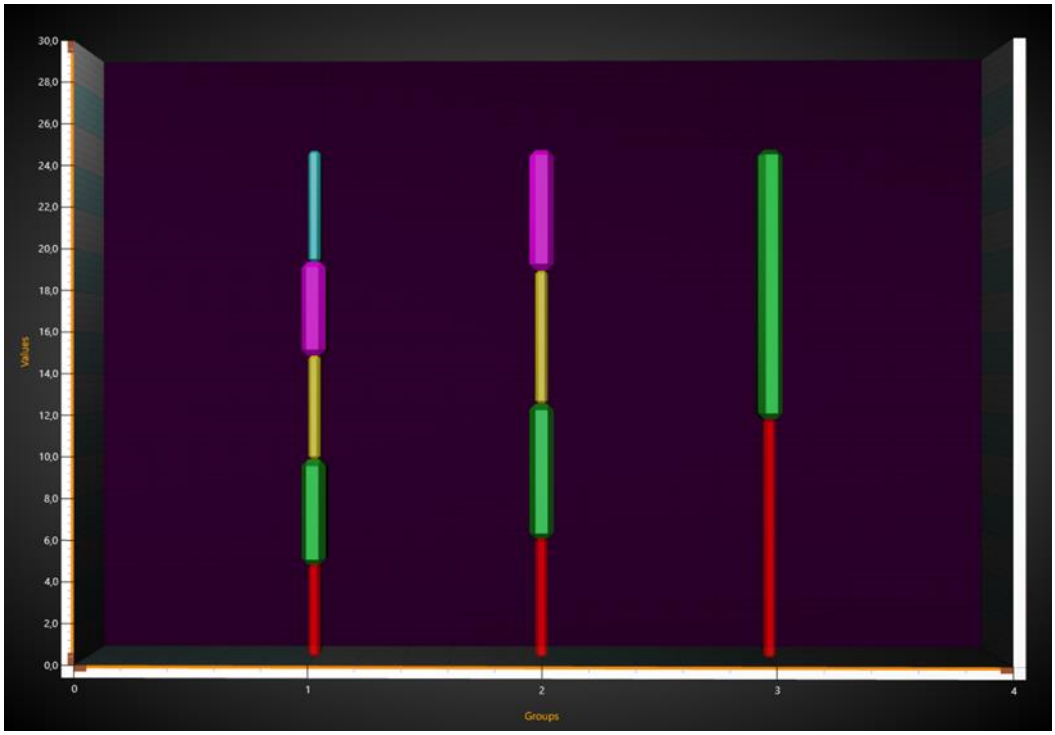


图 7-65. BarViewOptions.ViewGrouping = StackedStretchedToSum. 将所有具有相同 X 值的柱状进行堆叠，并拉伸至 StackSum；此例中为 25。

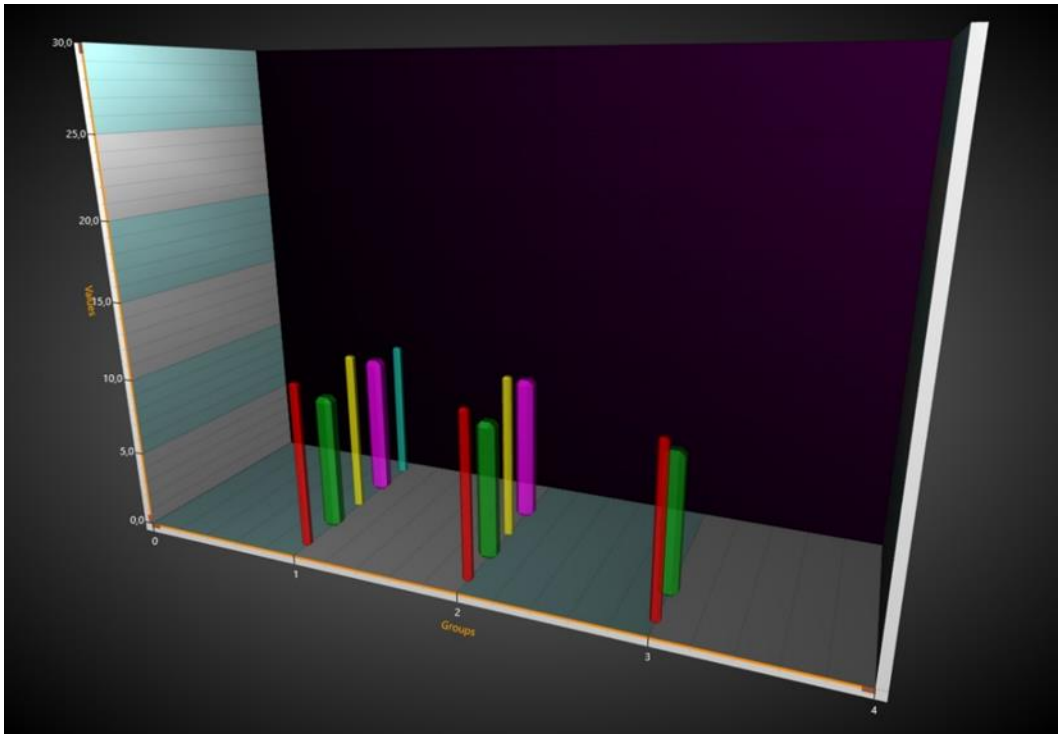


图 7-66. BarViewOptions.ViewGrouping = Manhattan. 第一个系列的值显示得最接近摄像机，最后一个系列显示得最远。柱状的 X 值控制着柱状在 X 维度的位置。

7.13.2 柱状样式

用 ***BarSeries3D*** 的 ***Shape*** 属性可控制柱状的形状。另外，对于某些形状，可以使用 ***CornerPercentage*** 来改变角的圆滑度，用 ***DetailLevel*** 改变视觉特性。

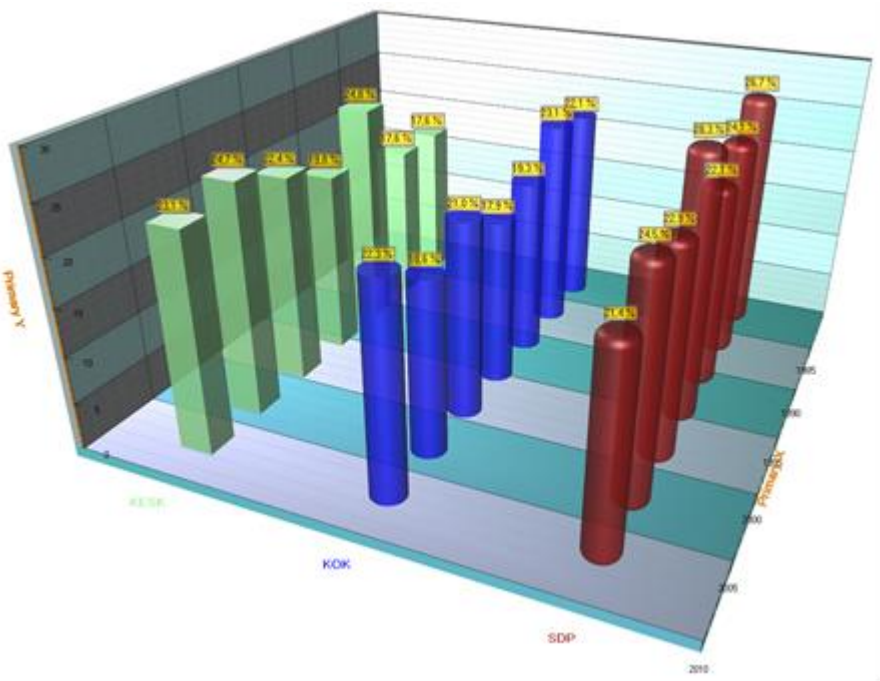


图 7-67. 柱状形状： Simple、Cylinder 和 RoundedCylinder.

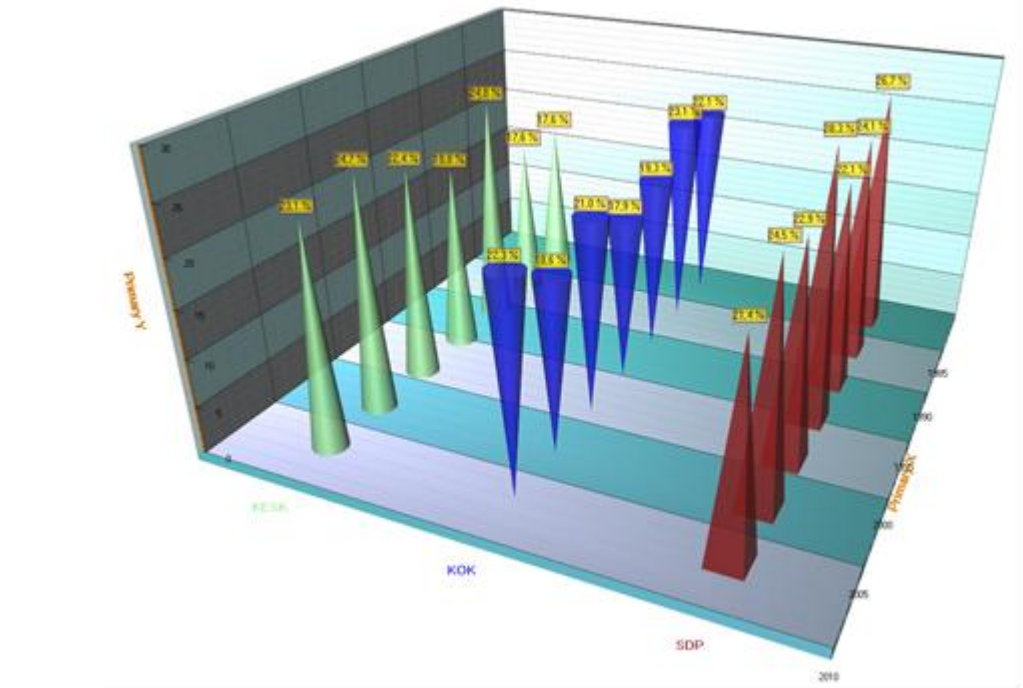


图 7-68. 柱状形状： Cone、 ReversedCone 和 Pyramid.

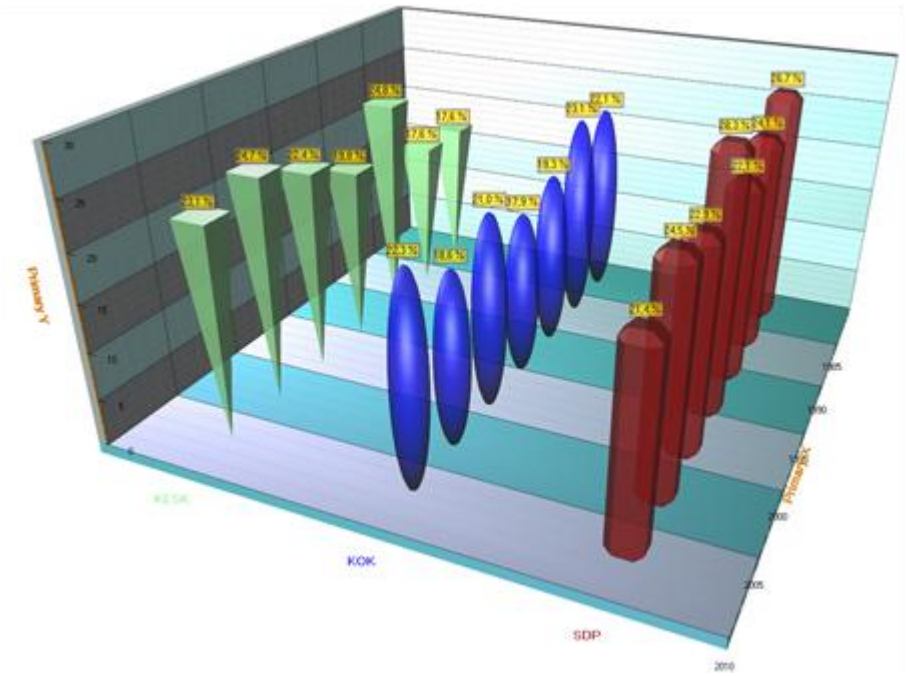


图 7-69. 柱状形状： ReversedPyramid、 Ellipsoid 和 Beveled.

7.13.3 设置柱状系列数据

柱状系列数据可添加为 ***BarSeriesValue3D*** -结构,包含 ***x***、***y***、 ***z*** 和 ***text*** 字段。

```
// 创建新的值数组
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, “”）;

// 向系列添加值
chart.View3D.BarSeries3D[0].AddValues（values, false）;
```

7.13.4 水平显示柱状

柱状以 Y 轴方向绘制。将摄像头旋转 90 度可垂直显示柱状。

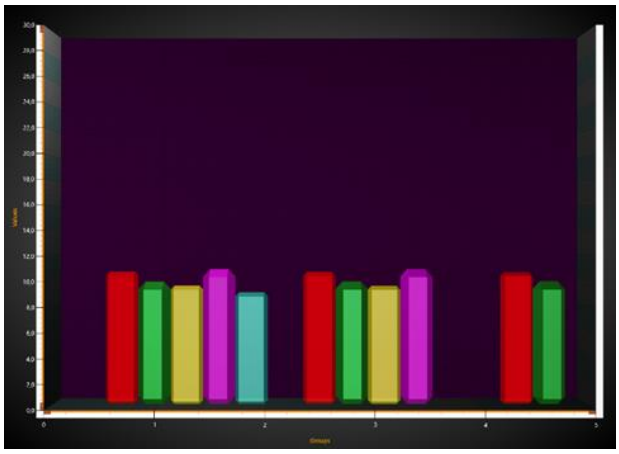


图 7-70. 垂直柱状视图

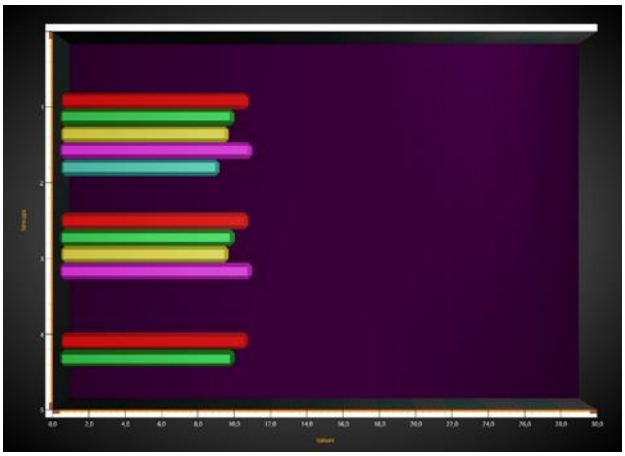


图 7-71. 水平柱状视图

以下是上文图片中设置垂直柱状视图的代码：

```
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();
```

以下是前文图片设置水平柱状视图的代码：

```
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();
```

7.14 MeshModels

演示示例： Vessels with sea depth; Mesh models coloring, wireframe; Mesh models realtime coloring; Mesh models by code

通过 ***MeshModels*** 属性列表，可以将 3D 模型从外部 3D 模型编辑器插入到 LightningChart View3D 中。 模型可以以 OBJ 格式导入，这是一种 3D 建模应用程序和游戏引擎中的通用格式。

注意! 由于 DirectX 11 不在支持 Direct3D X-格式文件（*.x），LightningChart v.7 v.7 之后不再支持该文件格式。

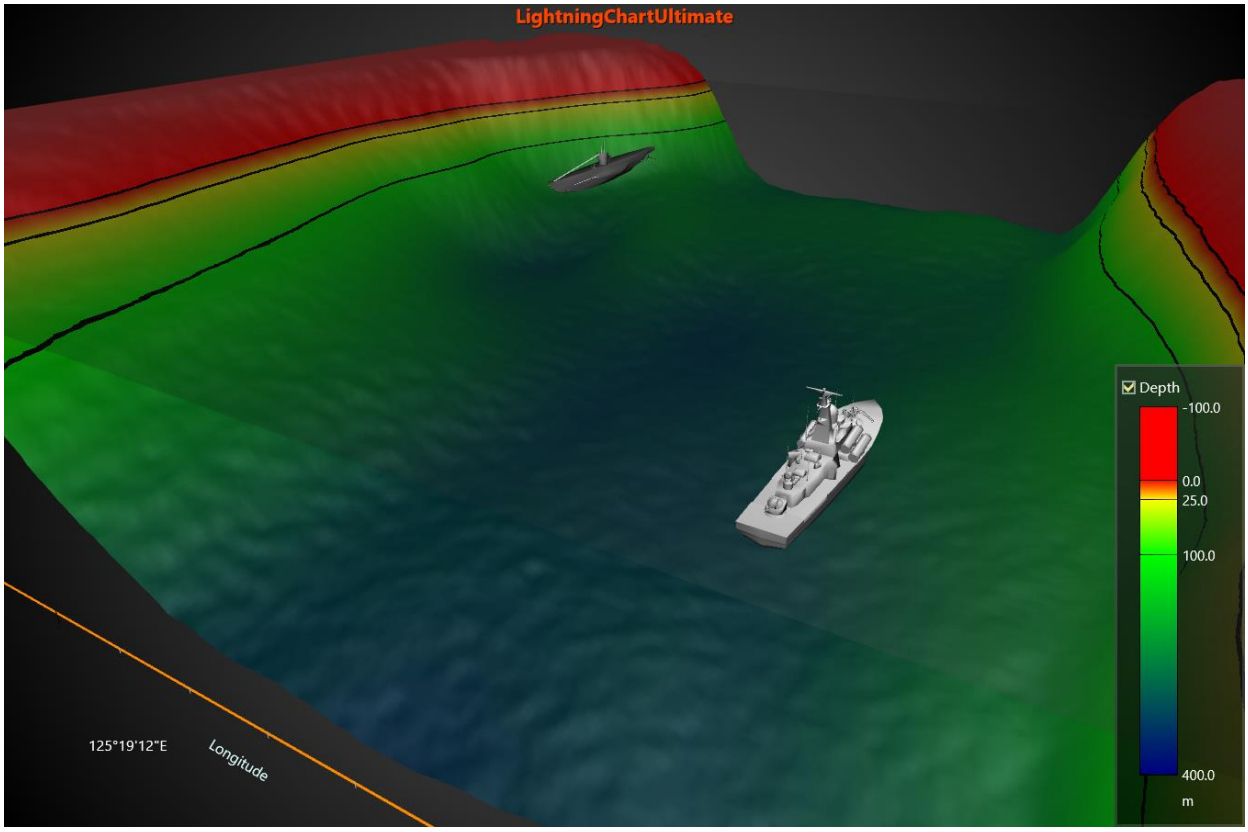


图 7-72. 在 View3D 视图中载入战舰和潜艇模型,，覆盖 SurfaceGridSeries3D 可视化海底深度数据。

7.14.1 加载模型

- 要从文件加载模型，可在 ***ModelFileName*** 属性内设置路径与文件名，或者使用 ***LoadFromFile*** 方法。当从文件加载模型，如果纹理填充存在于相同的路径中，并且可以访问 MTL 文件和图像文件，那么纹理填充也会被加载。
- 从 LightningChart 第 8.5 版本开始，***MeshModel*** 创建支持.obj 文件中的顶点颜色。顶点位置支持 Red, Green, Blue 以及 x、y 和 z 后的 Alpha 值（XYZRGBA）.
- 从数据流（stream）加载模型，可使用 ***LoadFromStream*** 方法。 数据流读取方法仅读取几何形状和材质，不读取纹理。
- 从资源（resource）加载模型，可使用 ***LoadFromResource*** 方法。

7.14.2 模型的定位、缩放和旋转

MeshModel 对象的 ***Position*** 依据其所分配到的 X、Y 和 Z 轴而定。模型可以通过编辑 ***Rotation*** 属性来旋转。其 ***Size*** 可用 ***Size*** 属性来定义，这是原始模型大小的一个要素集合，不遵循轴向范围或 3D 世界维度。

7.14.3 开启填充与线框

- 要显示填充，可设置 ***Fill*** = True
- 要显示线框，可设置 ***WireFrame*** = True, 并在 ***WireFrameLineColor*** 中设置首选线条颜色。

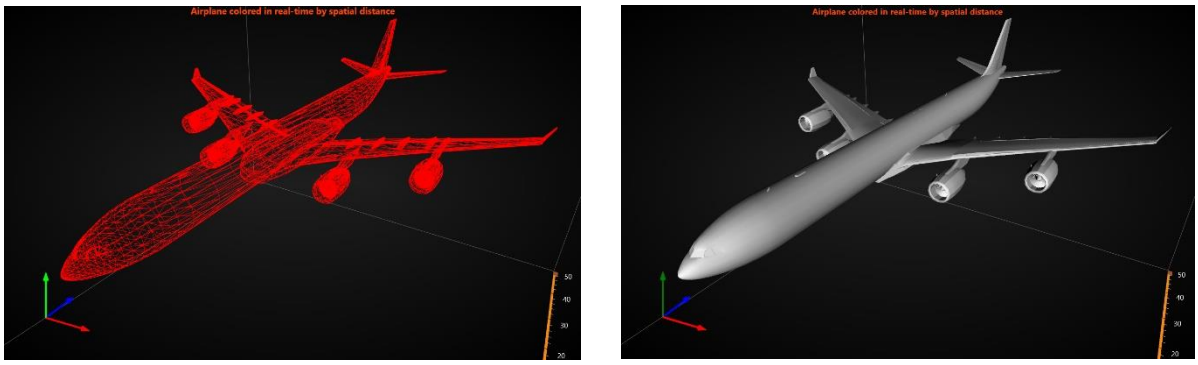


图 7-73. 飞机显示为线框（WireFrameLineColor = Red），默认以灰色填充

7.14.4 自定义着色填充

默认情况下，模型采用 OBJ 模型的颜色进行渲染。用 ***UpdateFillColors（int[] colors）***方法，可以为模型的顶点应用自定义颜色。此方法也可以定期调用，以应用实时颜色更新。***UpdateFillColors*** 方法需要一个与顶点位置长度（***X.Length***）相等的 ARGB 颜色数组。每个顶点一个颜色。

GeometryConstructed 事件报告顶点在轴值空间中的位置，如 ***X***、***Y*** 和 ***Z*** 数组。在应用着色时，例如通过其他图表对象（如数据点）的空间距离，尤其需要它们。当在初始化阶段时订阅 ***GeometryConstructed*** 事件处理程序，而当不在需要的时候可以取消。

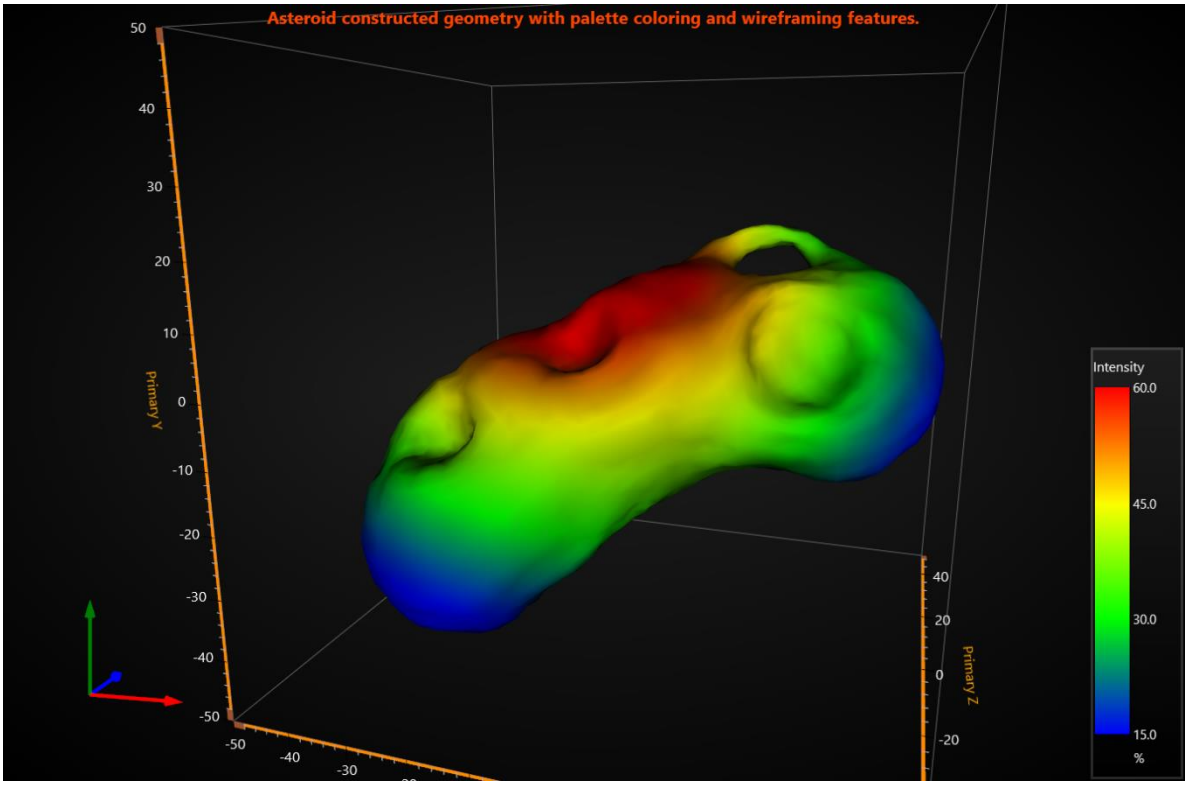


图 7-74. 用 UpdateFillColors 方法根据空间距离为 MeshModel 着色。

注意: ***ChartTools.ConvertDataToColorsByFixedIntervalPalette*** 方法可用来根据给定的调色板色阶来讲数据值转换为颜色 （ARGB int）。

7.14.5 自定义着色线框

线框可以采用自定义颜色进行着色。用 ***GeometryConstructed*** 事件处理器可获得所需的颜色数组长度，并用 ***UpdateWireframeColors*** 方法可应用新的颜色。

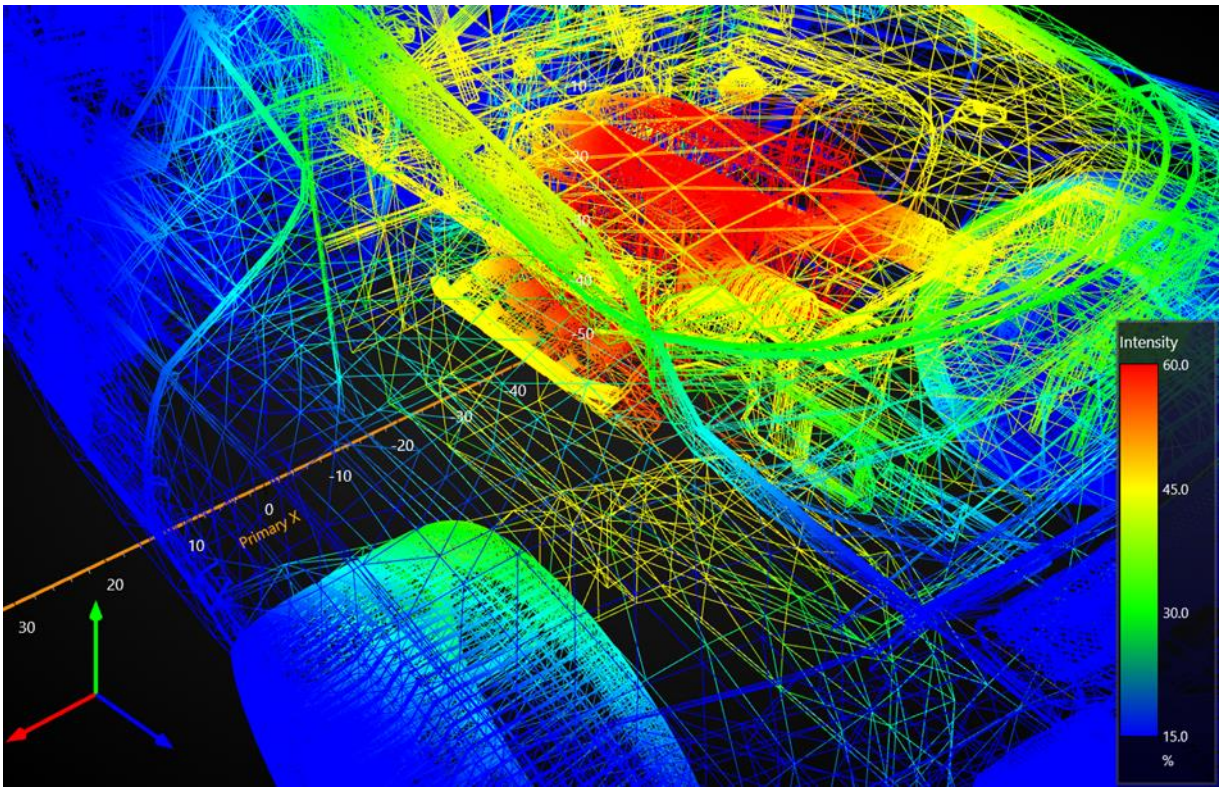


图 7-75. 用 UpdateWireframeColors 方法根据空间距离为 MeshModel 线框着色。

7.14.6 反向顶点缠绕顺序

有些模型是用反向缠绕顺序制作的，因此剔出会使它们不可见。假如一个模型不能正确显示，需将 ***Cull*** 设置在 ***Clockwise***、***CounterClockwise*** 与 ***None*** 之间进行变更。

```
meshModel.Cull = Cull.CounterClockwise;
```

7.14.7 Shade mode

可以控制灯光来影响 MeshModel 颜色。 将 ShadeMode 属性设置为 Flat，光照对模型没有影响。默认情况下，ShadeMode 设置为 Gouraud（光照影响模型）

```
model.ShadeMode = ShadeMode.Flat;
```

请注意，禁用照明效果会导致模型失去一些深度感知。

7.14.8 MeshModel 渲染顺序

在 LightningChart 第 8.5 版本中引进了 ***RenderingOrder*** -属性。该属性控制着 MeshModel 是否在例如 PointLineSeries3D 和 SurfaceGridSeries3D （***BeforeSeries***）等其他系列之前渲染，或是在他们（***AfterSeries***）之后渲染。具有类似 ***RenderingOrder*** -设置的 MeshModels 是按照它们添加到图表中的顺序绘制的。

```
meshModel.RenderingOrder = MeshModelRenderingOrder.BeforeSeries;
```

RenderingOrder 可以影响以上所有半透明的 MeshModels。其用途可确定是否可以通过模型看到其他系列。如果一个 MeshModel 使用不透明颜色，都会挡住后面所有的东西，无论 ***RenderingOrder*** -设置为什么。

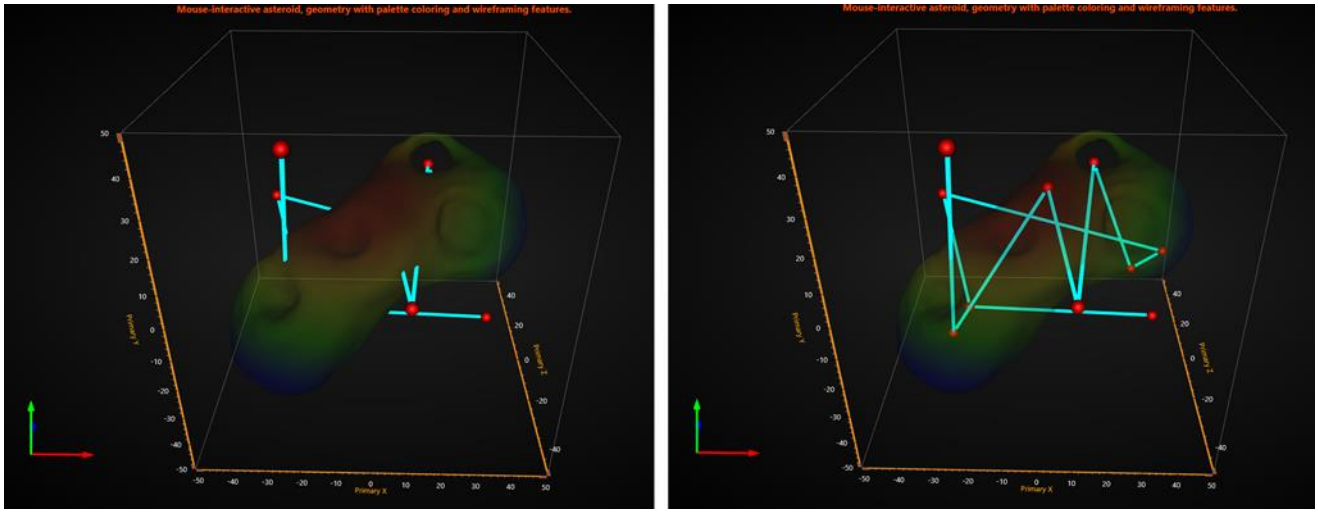


图 7-76. 一个半透明 MeshModel 的 RenderingOrder，左侧设置为 BeforeSeries，右侧设置为 AfterSeries 。 使用 BeforeSeries -选项，即使模型颜色是透明的，也无法通过模型看到如 PointLineSeries3D 等其他系列。

注意：当前的 Rectangles3D 和 Polygons3D 不受 ***RenderingOrder*** 的影响，因为它们不看作是系列。

7.14.9 从顶点以编程方式构造网格模型

从第 v.8.2 版本开始， ***MeshModel*** 支持以编程方式构造 ***MeshModel*** 几何形状。这可以实现对通过计算产生的对象和形状可视化。

可用的 ***Create*** 方法如下：

- *Create（positions, colors, indices）*
- *Create（positions, colors, normals, indices）*
- *Create（positions, textureCoordinates, bitmap, textureWrapMode, indices）*
- *Create（positions, normals, textureCoordinates, bitmap, textureWrapMode, indices）*

索引数组（***indices***）参数是可选的。如果提供有的话，他们将定义从给定的数组中使用哪些顶点、颜色、光法线和纹理坐标。当多个三角形共享相同的顶点时，使用索引可以节省资源。

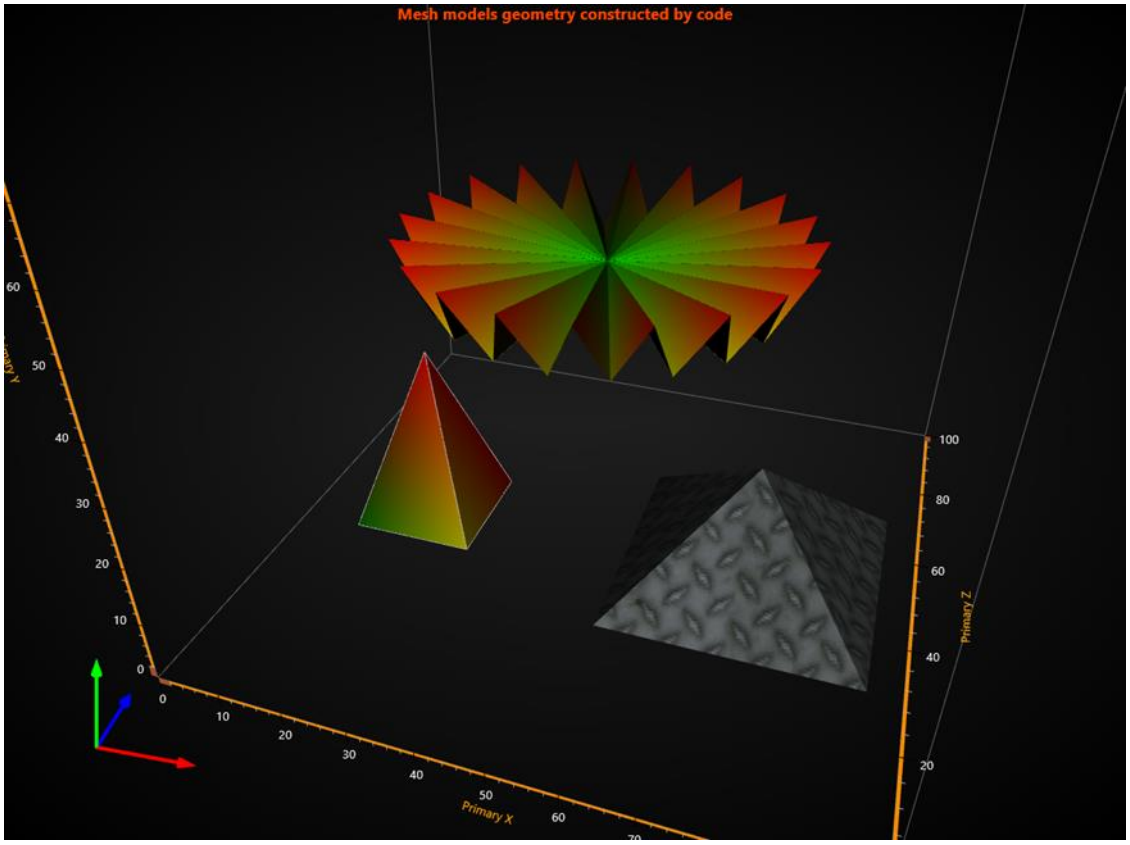


图 7-76. 通过代码构造 MeshModels

旋转、缩放和定位属性等以及事件也适用于从顶点以编程方式创建的 ***MeshModel***，与所加载对象的工作方式相似。

创建 MeshModels 的另一种方法是使用 CreateFromTriangles() 方法。 它基于给定的顶点数组 (PointFloat3D[]) 和颜色 (Color[] 或 int[]) 以及可选的法线数组 (PointFloat3D[]) 创建模型

7.14.9.1 有效地更新位图填充

当用 ***Create*** 方法创建了一个 ***MeshModel*** 后，可提供位图和纹理坐标作为参数，可以非常有效地更新位图而不用重建几何结构。调用 ***UpdateFillBitmap*** 方法可以进行更新。

注意! UpdateFillBitmap 方法不适用于从 OBJ 文件加载的模型。

7.14.10用鼠标跟踪模型

用 ***MeshModel*** 可进行基于三角形的鼠标位置跟踪。采用 ***MouseTriangleTraced*** 事件，可指示距离摄像头最近的三角形和鼠标位置。

事件参数有以下信息：

- ***IntersectionPointAxisValues***: 三角形面在坐标轴中的交点
- ***ModelSpaceTriangleCoordinates***: 3个三角形角（顶点）的数组，鼠标在3D模型空间坐标中点击

- ***WorldSpaceTriangleCoordinates***: 3个三角形角（顶点）的数组，鼠标在3D世界坐标系中点击
- ***NearestCoordinateIndex***: 追踪的三角形的最近坐标索引，值为0…2。用索引从***ModelSpaceTriangleCoordinates*** 或***WorldSpaceTriangleCoordinates***数组中提取索引。

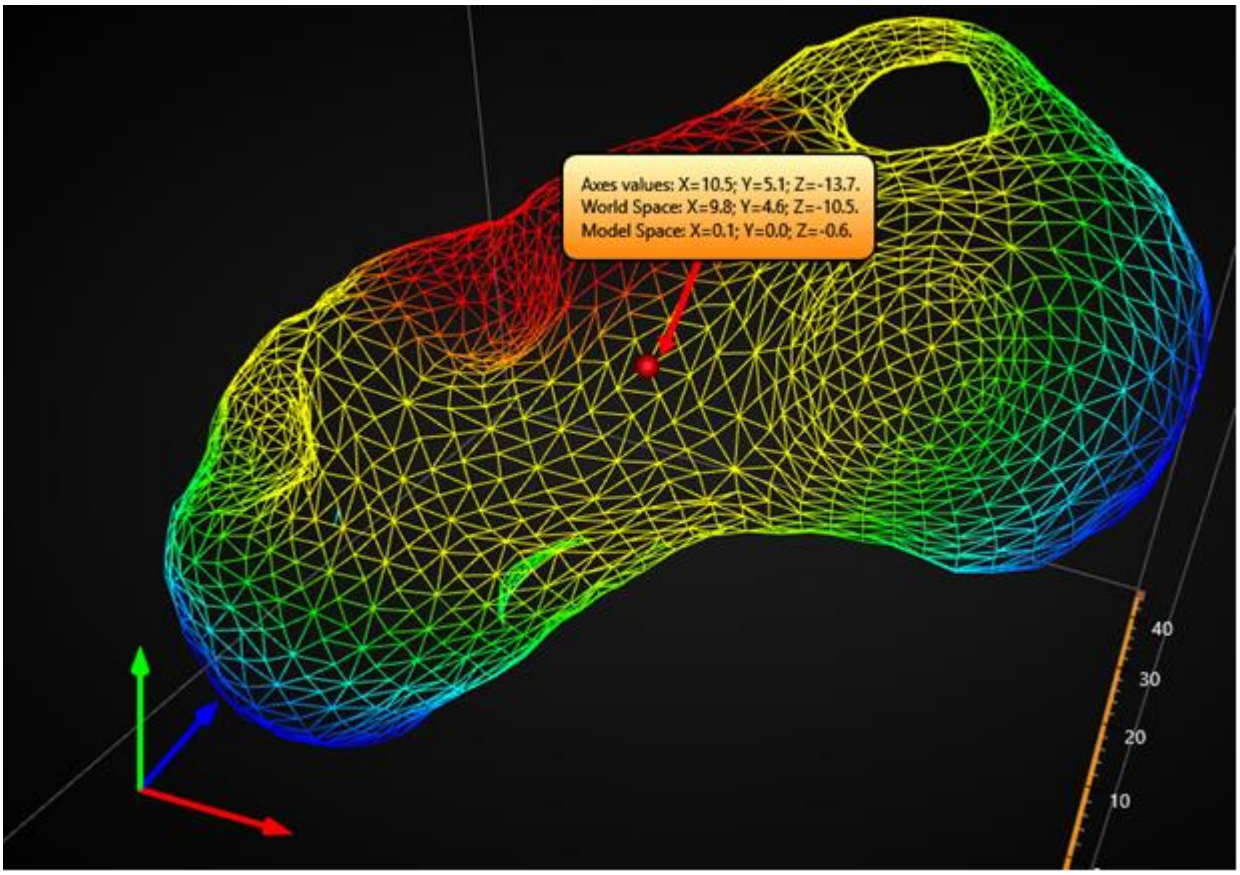


图 7-78. 用鼠标跟踪 MeshModels； 跟踪结果在注释框中显示。

7.15 VolumeModels

演示示例： Volume head; Volume flow; Volume geo; Volume skeleton; Volume wave interference

VolumeModels 方法用于通过 Direct Volume Rendering 实现体数据可视化。***VolumeModel*** 获取其中的体数据并将其可视化。LightningChart 的体渲染引擎是基于 **Volume Ray Casting**。

该算法通过沿着在数据集内移动的光线轨迹的体积数据采样来生成图像。**Volume Ray Casting** 的硬件加速的简单实现需要为体对象生成边界。通常，它们由一个立方体表示。该技术的主要优点是在没有伪像的情况下，具有较高的渲染质量，并且使用了可互换的射线功能。

RayFunction 是算法的核心，为其提供了非常高的灵活性。该一方法非常强大，因为它指定了数据采样和组合数据的方式。这使得它成为一个非常有用的特征提取工具。

注意! 只有当采用 DirectX 11 渲染时，VolumeModels 才可用。

7.15.1 加载数据

关于如何将数据导入到 ***VolumeModel*** 中有几种方法：

- 数据可以作为表示数据集切片的图像集提供给 ***Data*** 属性
- 数据可以通过多种方式直接提供给 ***VolumeModel*** 的构造函数
- 数据可以通过一个加载函数提供给 ***VolumeModel***

加载函数和构造函数允许将数据作为切片的集合（类似于 data 属性）提供，或者作为字符串提供切片所在文件夹的路径。数据也可以作为一种用我们的工具所创建的纹理地图提供。一个纹理图由切片组成，但是其补充也需要关于图片上的切片数量的更多信息。这是高效使用 GPU 输入缓冲器所必需的。纹理图可以通过 ChartTools.CreateMap 函数创建。直接导入纹理地图用来加快启动一个非常大的数据集的应用程序。

7.15.2 属性

VolumeModel 包含 LightningChart 中 3D 对象的典型属性，例如 ***Visible***、***Rotation***、***Size***、***Position***、***MouseInteraction*** 和 ***MouseHighLight***。此外，对象有一些特别的属性，可以规定 Volume Rendering 引擎如何处理它。

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

图 7-79. VolumeModels 属性树

7.15.3 射线功能

用 ***RayFunction*** 属性可以在 LightningChart Volume Rendering Engine 中的三种体素采样及组合方式中选择一种：

- ***RayFunction.Accumulation*** 可以收集并组合尽可能多的数据。这种技术产生的可视化效果看起来像是半透明的凝胶。下图展示了一个实现医疗数据集可视化的***RayFunction.Accumulation***应用程序示例。

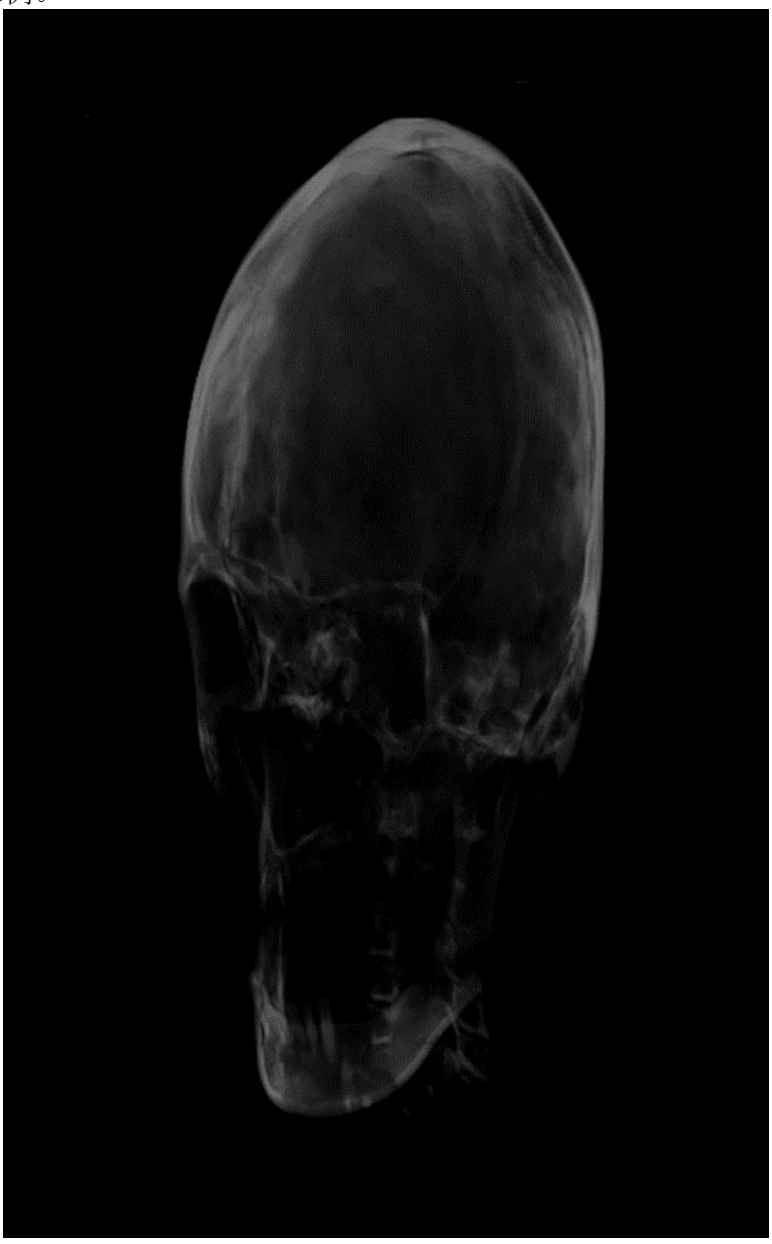


图 7-80. RayFunction.Accumulation 的医疗应用程序示例

- ***RayFunction.MaximalIntensity*** 只考虑射线采样的最亮值。在视觉上，它提供了一个与X射线图像非常类似的结果。通过它可以获得关于对象内部结构的更多信息。以下展示了用于骨架可视化和超声波干涉仿真的***RayFunction.MaximalIntensity***应用程序。

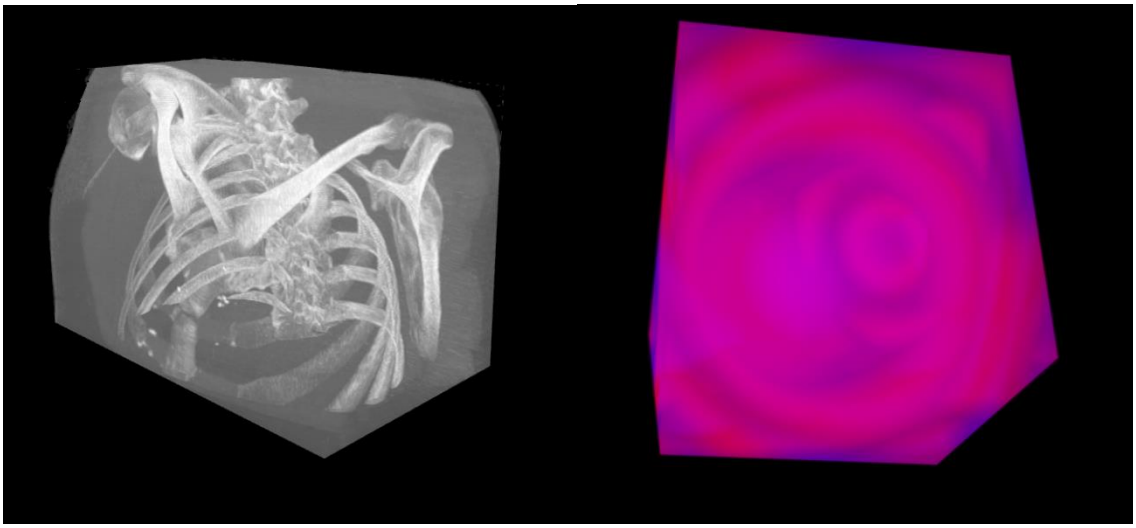


图 7-81.一个 Maximum Intensity Ray Function （最大强度射线功能）应用程序示例

- ***RayFunction.Isosurface*** 以一种类似多边形模型渲染的方式绘制模型表面。其结果与用 **Indirect Volume Rendering**生成的类似。 图片显示了人类颅骨CT可视化和水流模拟的 ***RayFunction.Isosurface***应用程序示例。

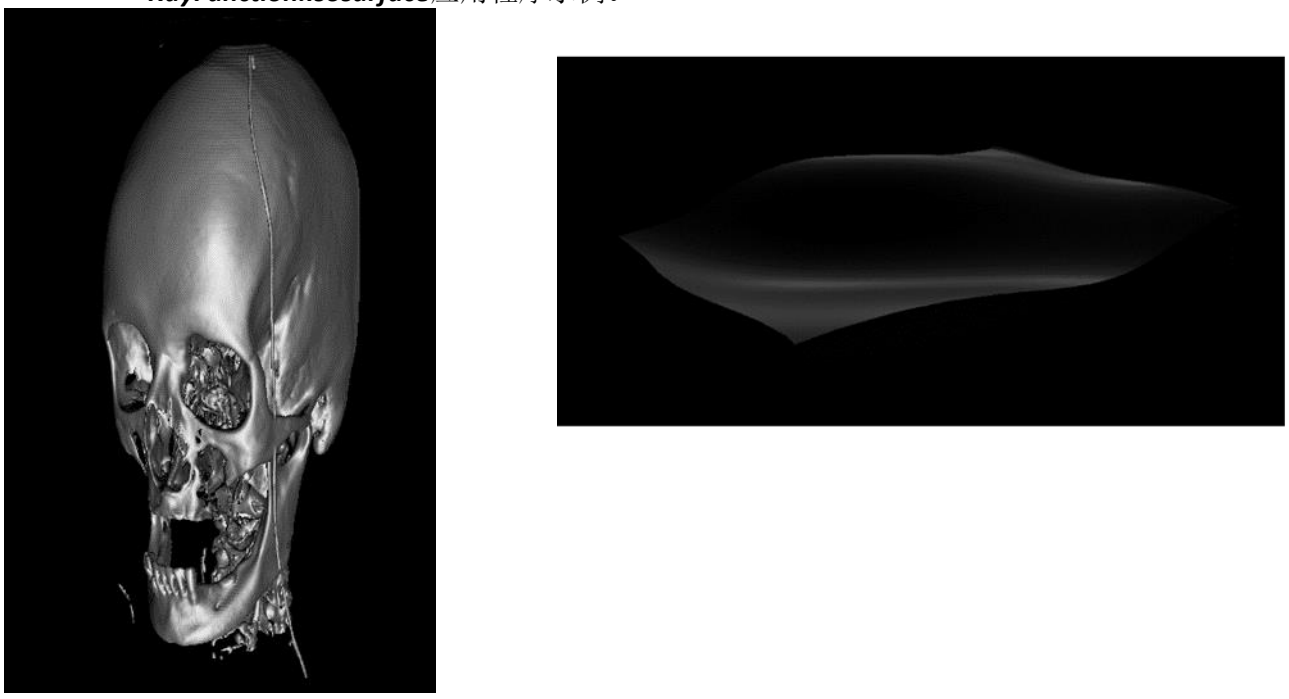


图 7-82. 一个 Isosurface Ray Function （等值面射线功能）应用程序示例

7.15.4 Threshold 阈值

Volume Rendering Engine 可以通过属性对 ***VolumeModel*** 应用一个阈值范围。每种颜色通道都有单独的边界。只有当相应的颜色值在所有通道中低于高界限，高于低界限时，体素才能直观的显示出来。符合条件的区域是不可见的。鼠标点击测试时不考虑此属性。

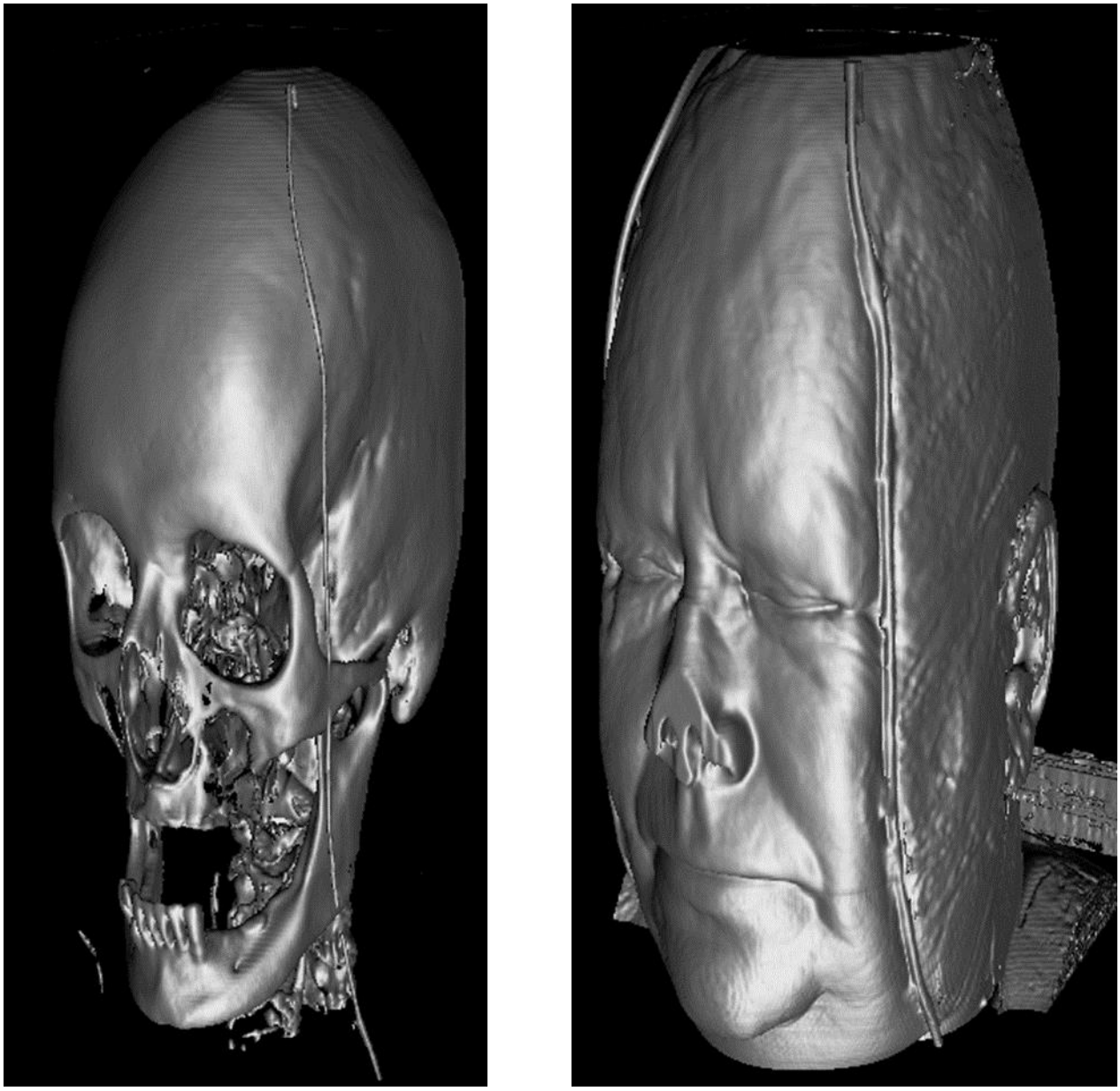


图 7-83. 两个不同阈值设定示例

7.15.5 Color clipping 颜色裁剪

演示示例：*Coloring Volume Model*

体积模型具有 ClipColorRange 和 ColorRangeToClip 属性，它们与阈值一样，可用于从模型中裁剪某些颜色。 但是，它们的工作方式相反。 颜色裁剪不会渲染定义的颜色范围内的颜色，而阈值会移除范围外的颜色。

ClipColorRange 控制是否启用颜色裁剪。 ColorRangeToClip 允许设置应删除的实际颜色范围。 可以通过每个颜色通道的 Min 和 Max 属性分别设置最小和最大裁剪值。 或者，可以通过将 RangeRGB 对象

分配给 ColorRangeToClip 来同时修改所有值。 裁剪值应介于 0 和 1 之间，其中 0 表示颜色值 0，1 表示颜色值 255。设置范围后，将裁剪定义范围内的每个颜色组合。 裁剪同时考虑所有颜色通道。

```
// Enabling color clipping.
_chart.View3D.VolumeModels[0].ClipColorRange = true;

// Modifying a single channel value.
_chart.View3D.VolumeModels[0].ColorRangeToClip.Min.R = 0.1;

// Assigning all clip ranges simultaneously.
_chart.View3D.VolumeModels[0].ColorRangeToClip =
new RangeRGB(new PointRGB(0, 0, 0), new PointRGB(0.2, 0.2, 0.2));
```

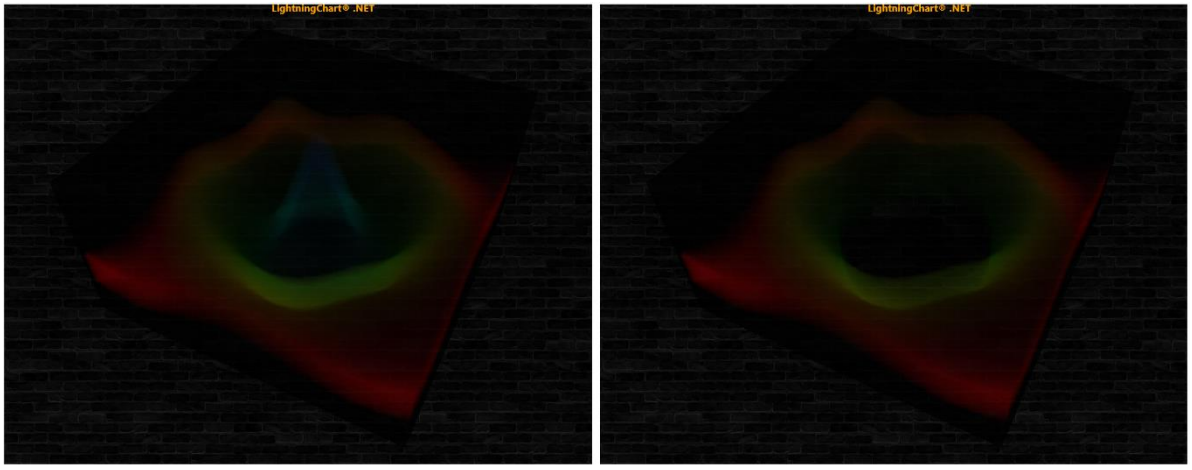


Figure 7-19. 左侧为原始体积模型。 右侧颜色剪裁用于移除蓝色通道。

7.15.6 Slice Range （切片范围）

用 ***SliceRange*** 属性可以切去 ***VolumeModel*** 的一部分。这对于探测对象的内部结构是一个很有用的方法。***SliceRange*** 包含有连个界限： ***Min*** 和 ***Max***, 二者均由三个指向（对准）的浮点值表示。

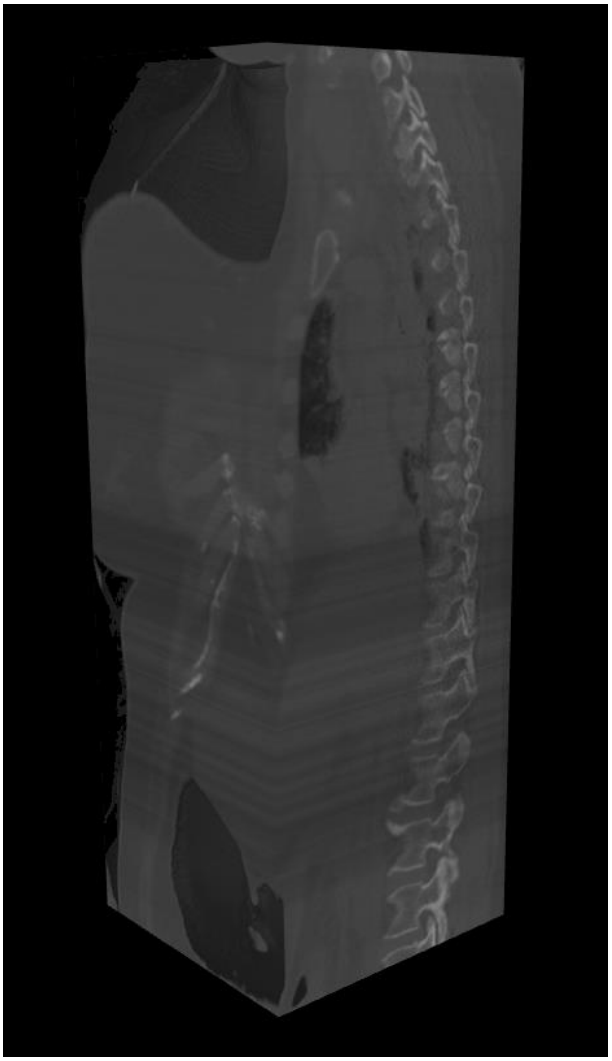


图 7-84. Accumulation Ray Function 与 SliceRange 修改示例

7.15.7 Sampling Rate Options 采样率选项

SamplingRate 是最终图像质量的一个非常重要的属性。它定义了沿着射线的轨迹对体数据集进行采样的频率。较高的 ***SamplingRate*** 可以生成更好的质量，但是也需要更强的硬件性能。***SamplingRate*** 影响着 ***RayFunction*** 选项，特别是 ***Accumulation***。当使用 ***Maximal Intensity*** 时，由低采样率产生的伪影更不明显。此外，在非常高的采样率情况下，***Isosurface*** （等值面）可能会非常清晰。通常，最佳点等于沿着射线轨迹放置在一侧的体素的数量。

SamplingRateOptions 包含有用于 ***SamplingRateManager*** 设置的几个选项。***SamplingRateManager*** 用来达到硬件质量和帧速率之间的最佳平衡。默认情况下，通过将其属性 ***Enabled*** 设置为 ***true*** 来打开 ***SamplingRateManager*** 。若设置为 ***false***, 则可以利用 ***ManualSamplingRate*** 值。 ***SamplingRateRange*** 可以定义 ***SamplingRateManager*** 的界限。 ***Inertness*** 规定了在性能发生变化时采样率的反应有多快。***TargetFPS*** 是一个目标值，即采样率管理器试图达到的值。

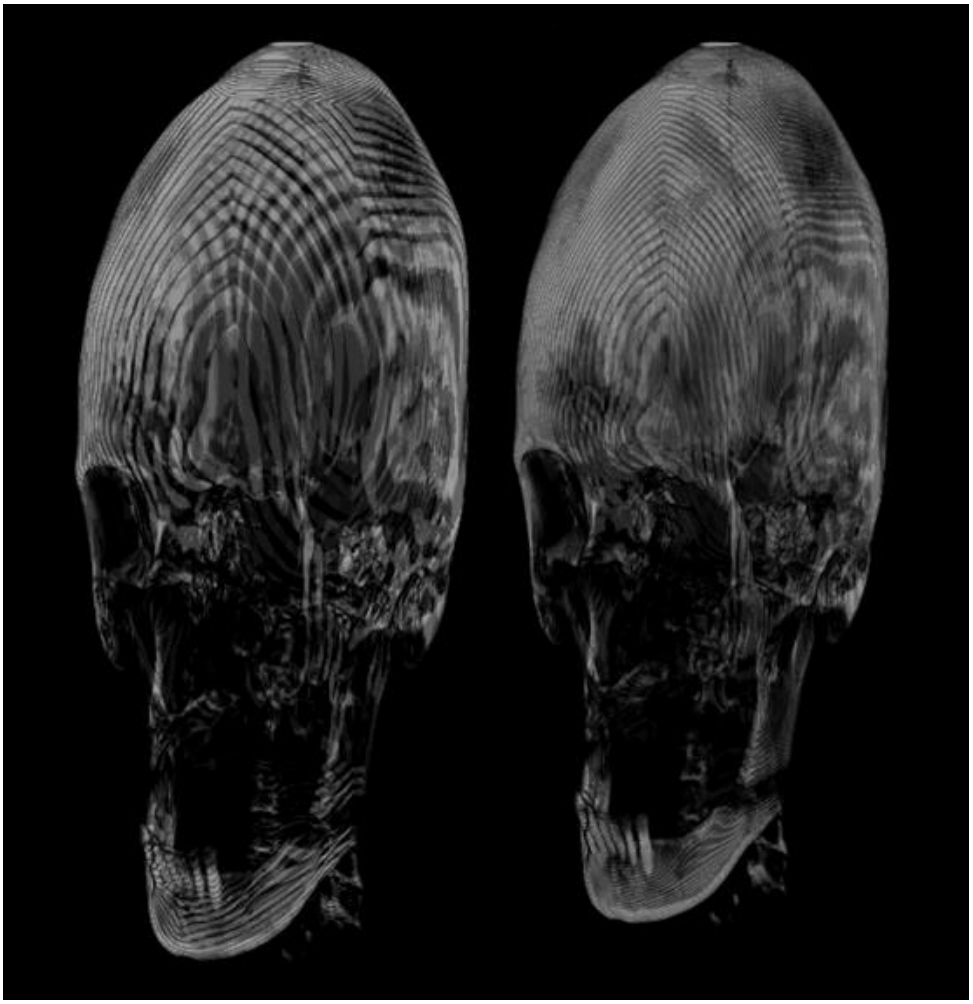


图 7-85. 低采样率示例: 32（左）, 64（右）

7.15.8 平滑度

Smoothness 属性可防止表面出现过高的细节化程度。这可以让模型表面更平滑，并进行降噪，减少过多的伪影。

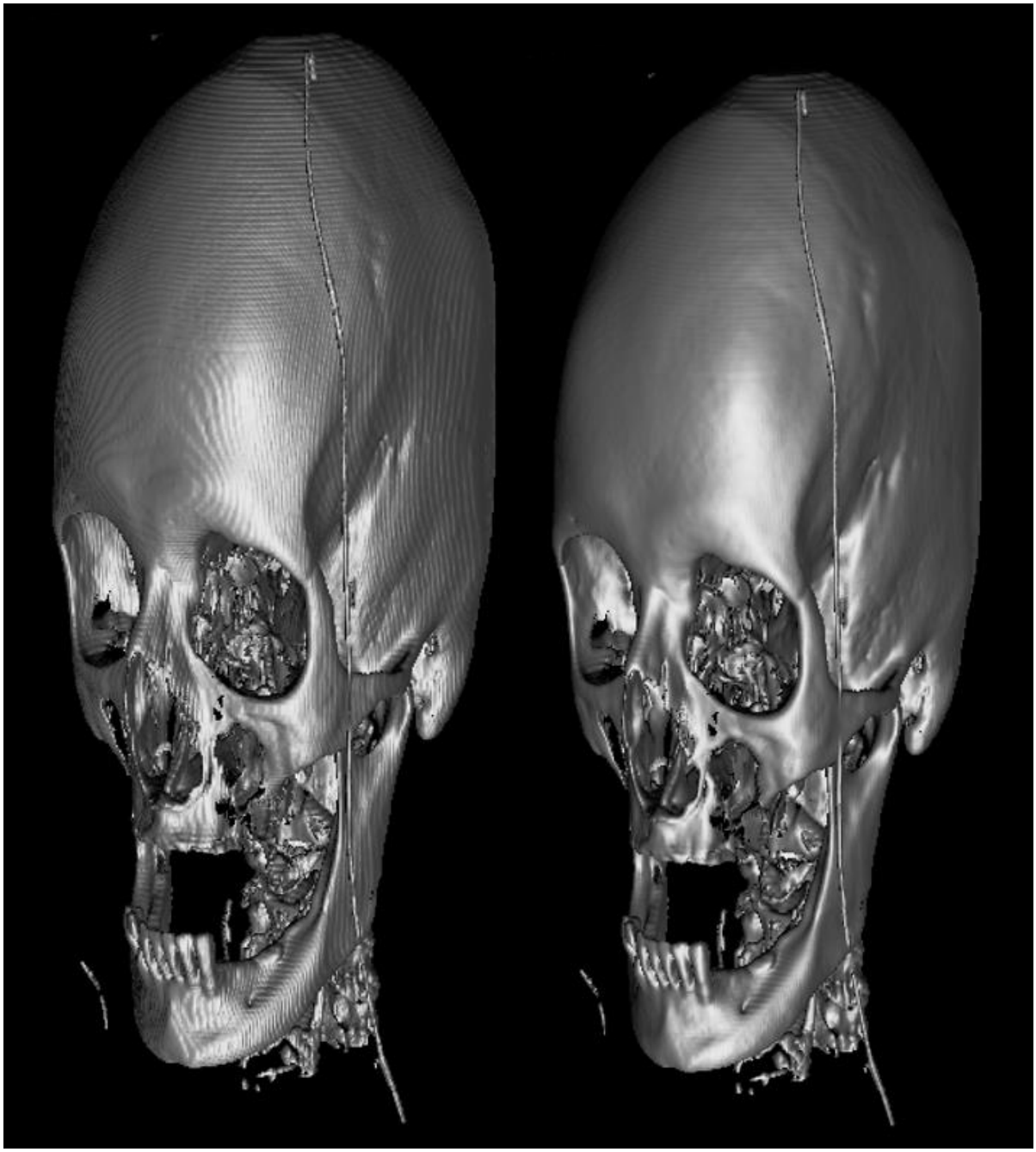


图 7-86. 过高采样率示例，经平滑度属性修整。

7.15.9 EmptySpaceSkipping

EmptySpaceSkipping 属性可定义空白空间的分辨率，跳过采样。***EmptySpaceSkipping*** 设定一个值低（16-32），可提高性能，但会导致在模型边缘产生伪影。

图 7-87. EmptySpaceSkipping 属性值过低示例

7.15.10 Opacity （不透明度）

Opacity 规定了 ***RayFunction*** 的 ***Accumulation*** 选项的性能。***Opacity*** 越低，对象将越透明。

图 7-88. Accumulation Ray Function Opacity 修改示例: 15%（左）, 45（右）

7.15.11 亮与暗

这些属性定义了图像的传递函数。每个变化都有自己的传递函数。可由线性函数表示为：

$$output = Brigthness * input - Darkness$$

7.16 Rectangle3D objects

演示示例： Rectangles/Planes; Surface mouse control; Parallel coordinates chart

用 ***Rectangle3D*** 可以呈现一个矩形，并调整至任何视角、任何大小、任何位置。这类对象可以添加到 ***View3D.Rectangles*** 列表中。通过依据 ***View3D.Dimensions*** 来定义矩形的尺寸大小还可以将其充当成平面。

在 3D 世界维度中以 ***Width*** 和 ***Height*** 设置 ***Size*** （而非 X、Y 或 Z 轴值）。通过 X、Y 和 Z 轴值来定义 ***Center*** 属性，以此设置中心点。 ***Rotation*** 属性规定了以度数为单位的旋转效果。

通过 ***Fill*** 属性可以修改填充设置。可以进行纯色和位图填充。若要采用位图填充，可在 ***Image*** 中设置位图，并开启 ***UseImage***。当设置 ***Fill.Layout = Stretch*** 后，位图可以拉伸来填充矩形。若设置 ***Fill.Layout = Tile***, 可以平铺此位图来填充矩形。使用 Fit 选项，位图填充指定区域，同时保持原始纵横比。 平铺量可以通过 ***Fill.TileCountWidth*** 和 ***Fill.TileCountHeight*** 属性来改变。

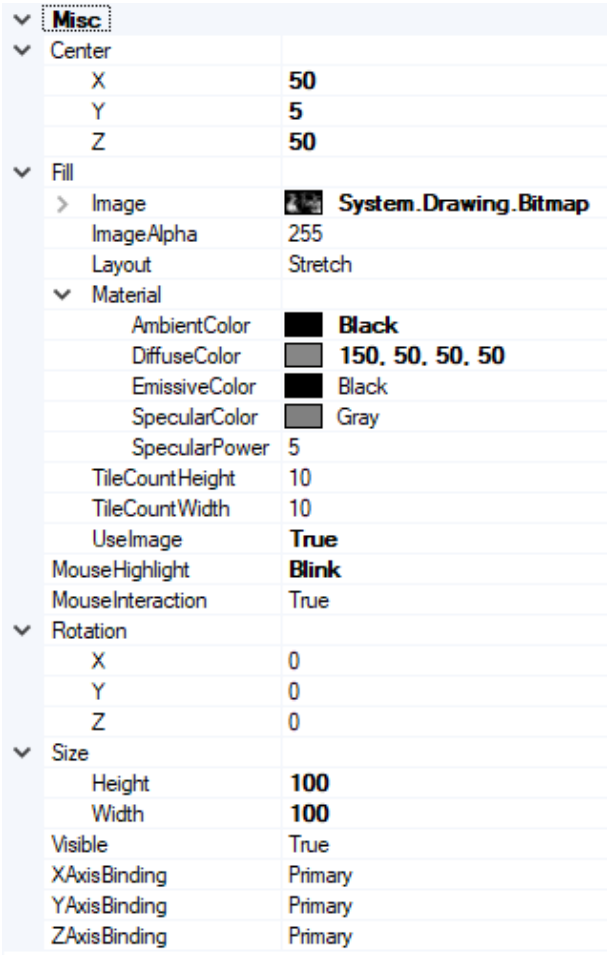


图 7-89. Rectangle3D 对象属性。

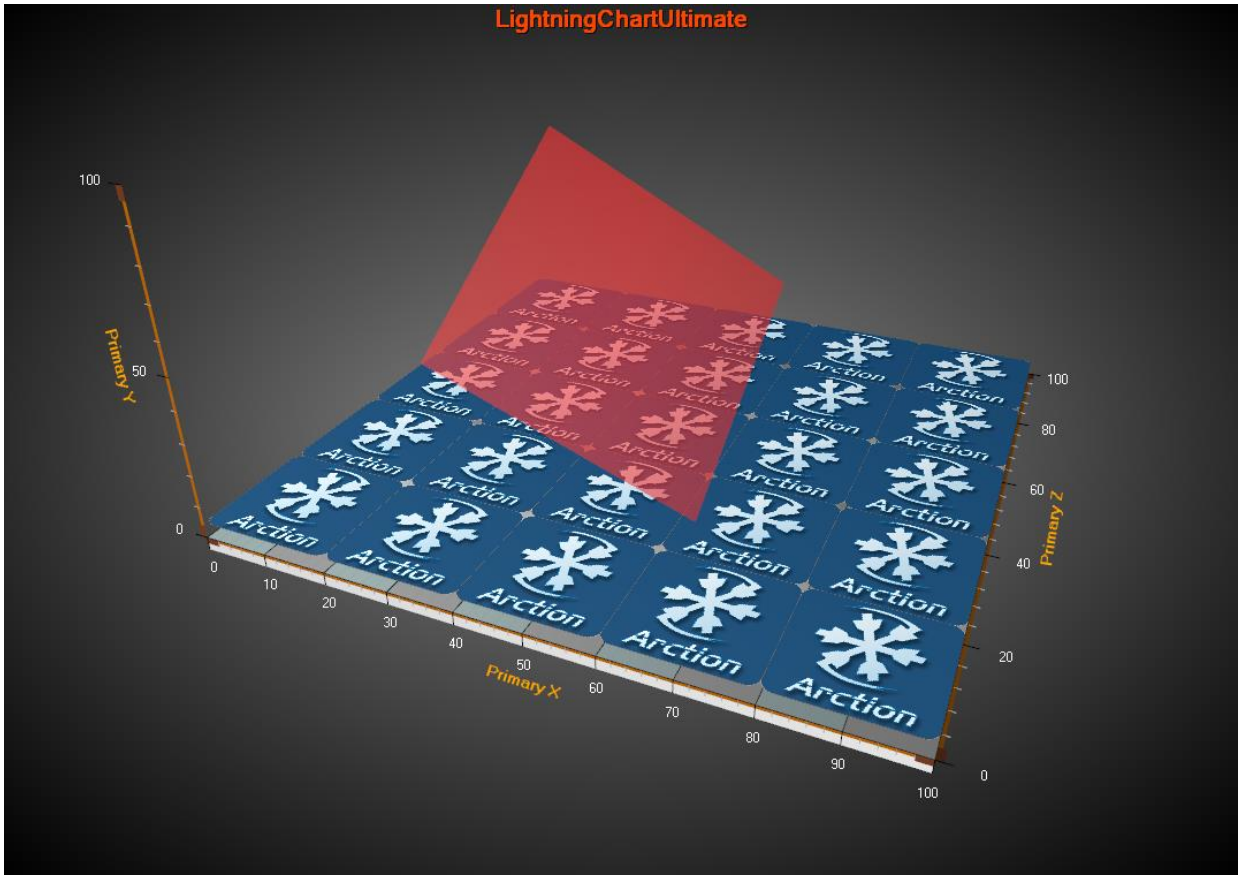


图 7-90. View3D 视图中的两例 Rectangle3D 对象。底部蓝色显示的位图填充设置为 Layout = Tile；上方经过旋转的红色矩形设置为透明色。

7.17 Polygon3D 对象

演示示例： 3D polygons; World population

Polygon3D 对象可以展现一个 2D 多边形，并拉伸至给定的 Y 值范围。这类对象可以添加到 ***View3D.Polygons*** 列表。

以 X 和 Z 轴值来定义多边形路径，并存储至 ***Points*** 数组中。用 ***YMin*** 和 ***YMax*** 值可设置 Y 值域。

Material.Diffuse 控制着矩形的主色调。用 ***Rotation.X***、***Rotation.Y*** 和 ***Rotation.Z***，以度数为单位可以将多边形旋转至其他角度。

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

图 7-91. Polygon3D 对象的属性。

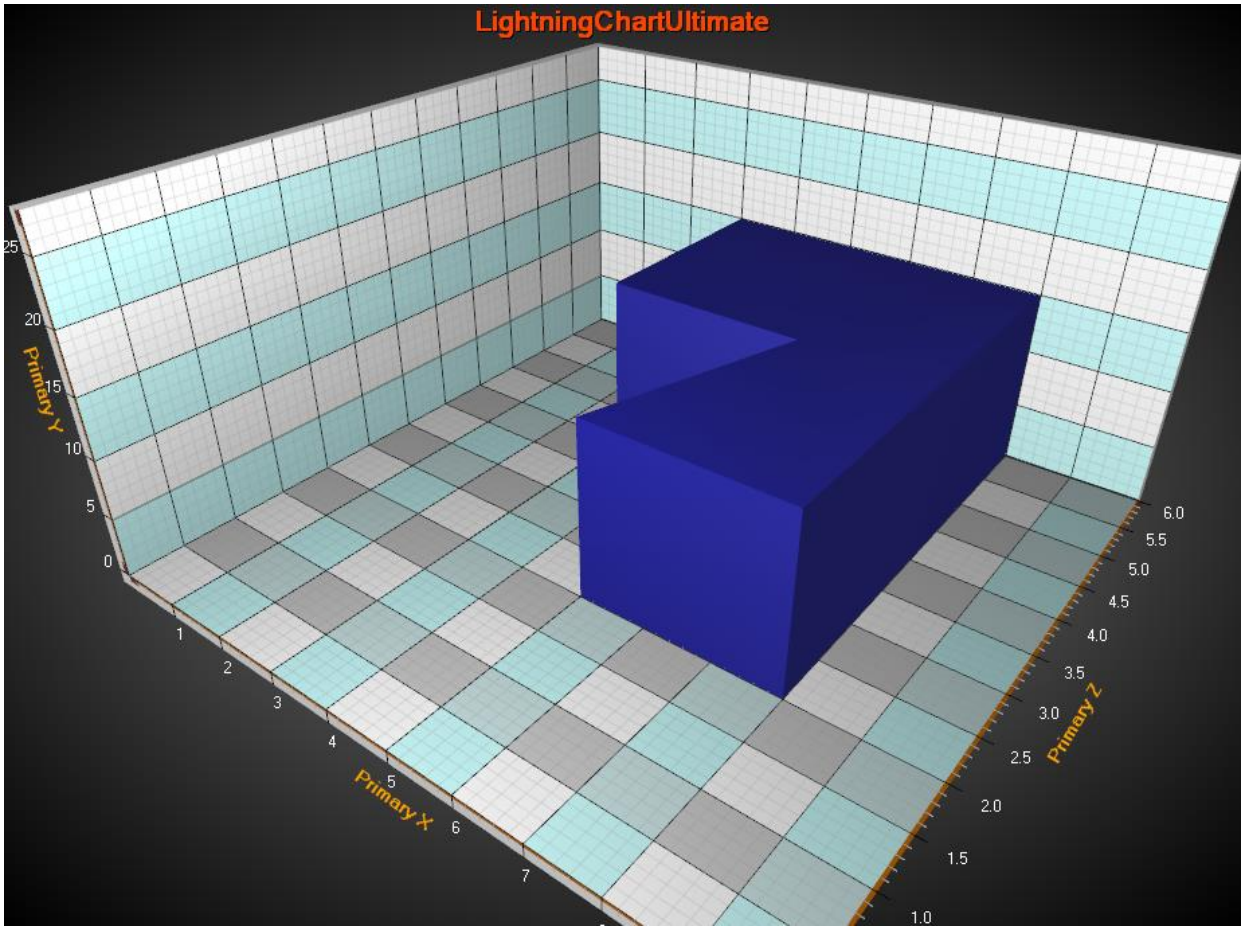


图 7-92. 一个 6 点多边形，范围 YMin = 0, YMax = 15.

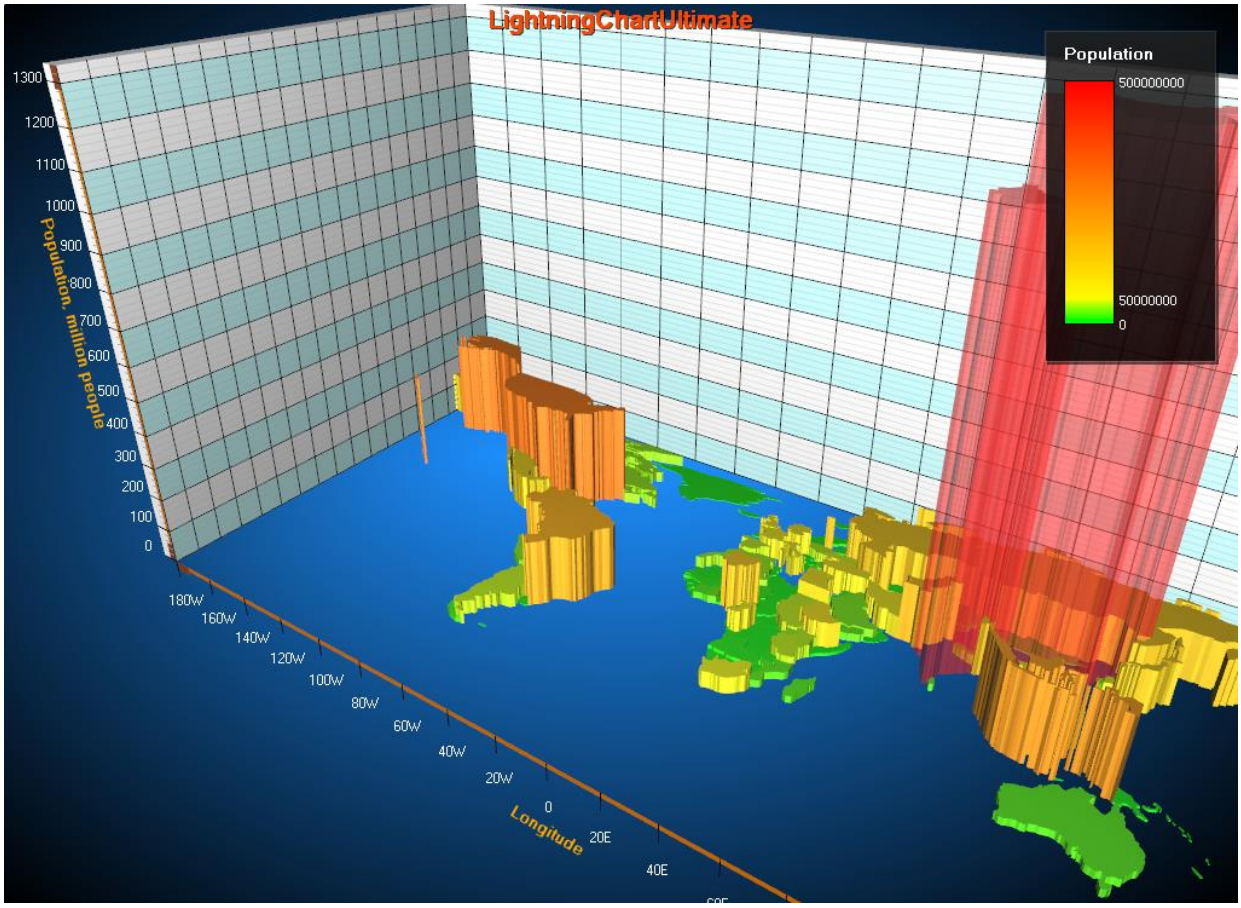


图 7-93. 用 Polygon3D 对象显示的世界人口图。根据地图数据的每个区域绘制一个 Polygon3D 对象。根据一个国家的人口总量来为该多边形着色，并设置其 YMax 值。由于中国和印度的人口量非常大，所以将其设置为透明色显示。

7.18 数据游标

从版本 10.5 开始，View3D 具有内置数据光标，它会自动跟踪最接近鼠标光标的系列值并将其显示在结果表中。光标由三个轴的十字线、最近数据值位置处的跟踪点、显示轴刻度上当前值的指示器以及结果表组成，其中除了轴的值之外还显示系列名称及其颜色。

可以通过 Visible 属性启用或禁用数据游标。还可以通过设置 ShowHaircrossLines 或 ShowLabels 或其他相应的“显示”属性（基于应隐藏的内容 false）来单独隐藏某些光标组件，例如线或轴的指示器标签。 IndicatorLength 和 IndicatorWidth 修改轴指示器，TrackingPointStyle 允许更改跟踪点，而 LineStyle 更改头发交叉线。 Results 属性包含修改结果表的所有选项。 轴标签的外观可以在配置类别下修改。

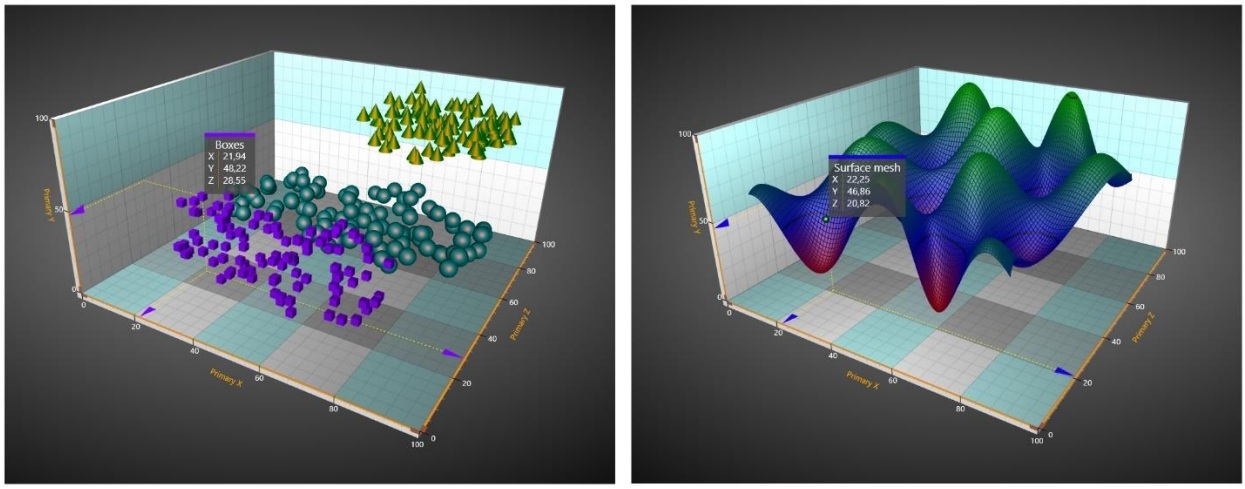
```
   // Enables data cursor but hides its axis indicator labels.
   _chart.View3D.DataCursor.Visible = true;
   _chart.View3D.DataCursor.ShowLabels = false;
   // Modifying the result table.
   _chart.View3D.DataCursor.ShowResultTable = true;
_chart.View3D.DataCursor.Results.Background.Color = Colors.DarkBlue;
```

数据游标会根据跟踪的系列是否具有可见线或仅具有可见数据点（散点图）来更改其行为。 如果该线可见，则光标会找到距离光标当前位置最近的系列及其值。 如果没有可见的线，则光标会在任何方向上追踪最近的数据点。 启用 SnapToNearestDataPoint 会覆盖此设置，使光标始终在任何方向上找到最近的实际数据点值。

数据游标适用于每个 View3D 系列和对象，但 Rectangle3D 和 VolumeModels 除外。

DataCursor	
Configuration	
XAxisArrowBackground	White
XAxisArrowUseSeriesColor	True
XAxisLabelVisible	True
XAxisLineVisible	True
YAxisArrowBackground	White
YAxisArrowUseSeriesColor	True
YAxisLabelVisible	True
YAxisLineVisible	True
ZAxisArrowBackground	White
ZAxisArrowUseSeriesColor	True
ZAxisLabelVisible	True
ZAxisLineVisible	True
IndicatorLength	5
IndicatorWidth	2,5
LabelFont	Segoe UI, 12pt
LabelIndicatorInside	True
LineStyle	
OffsetFromWall	0,55
RealTimeTracking	False
Results	
ShowColorIndicator	True
ShowHaircrossLines	True
ShowLabels	True
ShowResultTable	True
ShowTag	False
ShowTrackingPoint	True
SnapToNearestDataPoint	False
strTag	Tag
TrackingPointStyle	
TrackSeries	True
TrackSeriesPixelTolerance	20
Visible	True

图 7-96. 数据游标的属性树.



. 图 7 97. 具有 PointLineSeries3D 和 SurfaceMesh3D 系列的数据游标

7.19 缩放、平移与旋转

ZoomPanOptions 属性可用于管控缩放、平移与旋转设置。

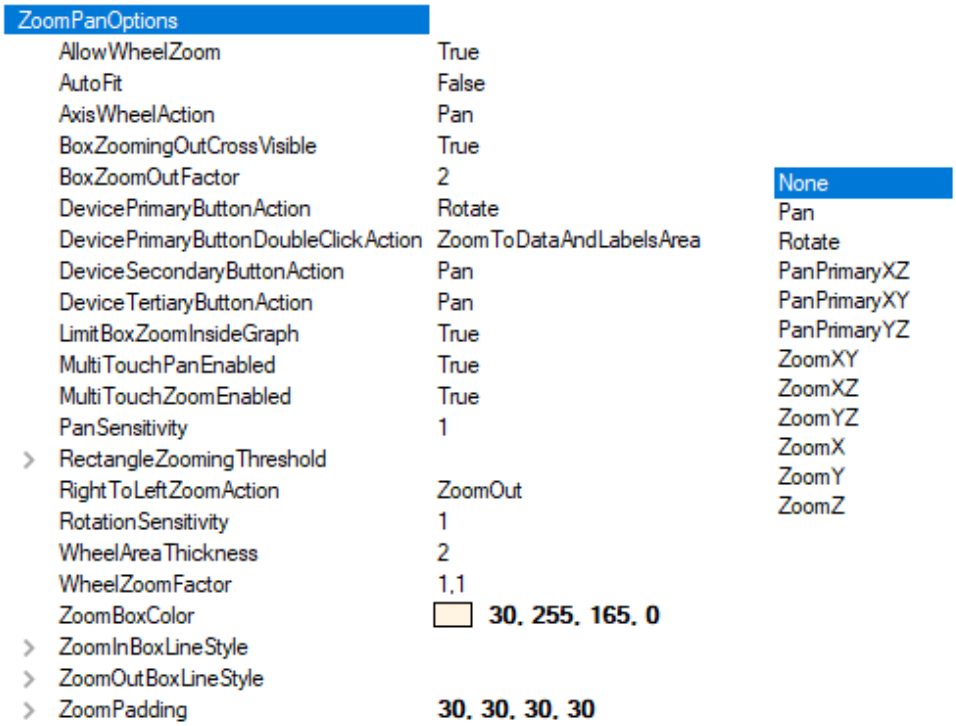


图 7-94. ZoomPanOptions 属性及子属性, 右侧为 LeftMouseButton / MiddleMouseButtonAction / RightMouseButtonAction 选项

根据设置，可以用鼠标滚轮、触屏捏/展、或通过在选定的 3D 平面上绘制一个框体，来进行缩放。对鼠标左键、中键或右键进行配置后可以进行平移、框缩放以及旋转。整个 3D 图表都可以进行平移，或者在保持 3D 场景位置不变的情况下调整主轴。

7.19.1 鼠标滚轮缩放

向上滚动鼠标滑轮可以放大，向下则缩小。用 ***WheelZoomFactor*** 可以调整每个鼠标滚轮事件所应用的缩放量。要禁用鼠标滚轮缩放，可以设置 ***AllowWheelZoom*** 为 ***false***。默认情况下，该属性设置为 ***true***。

7.19.2 框缩放

若要启用框缩放，可将框缩放分配给一个鼠标按钮操作属性。例如，设置 ***DevicePrimaryButtonAction = ZoomXZ*** 可将框缩放应用于 XZ 平面，而 Y 维度不受影响。分别对 ***ZoomXZ*** 或 ***ZoomYZ*** 进行设置可缩放其他平面。

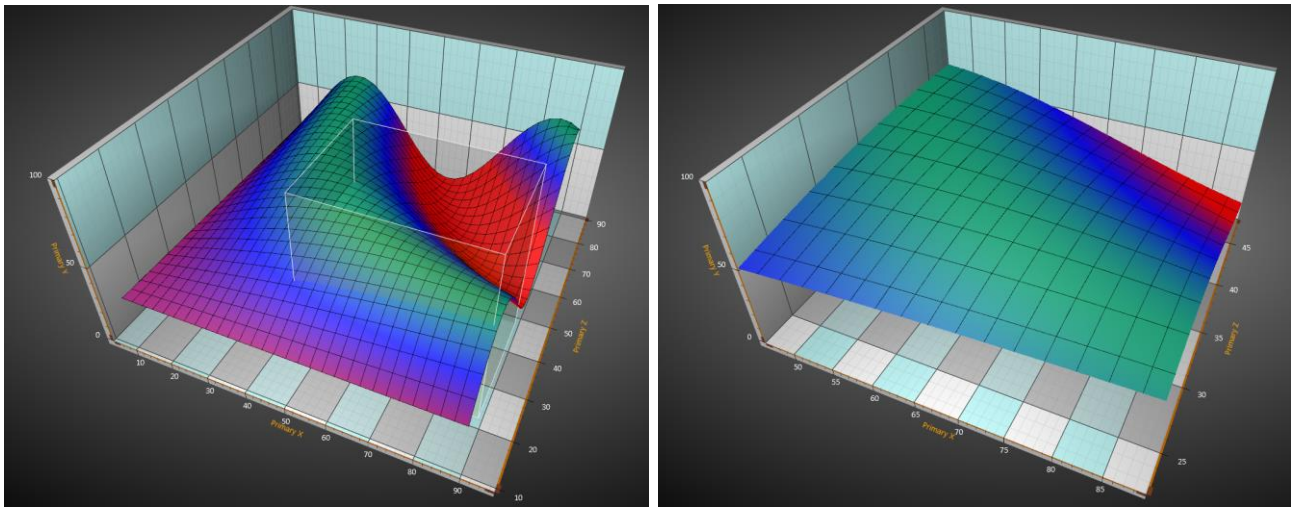


图 7-95. 左侧，正在进行 XZ-平面框缩放。右侧为缩放结果。X 和 Z 轴值域变更，而 Y 轴值域保持不变。

从左向右拖动框体可放大。缩放的值域范围应用于与所选平面相关的轴。

要仅对特定的 X、Y 或 Z 维度进行缩放，可选择 ***ZoomX***、***ZoomY*** 或 ***ZoomZ***。

从右向左拖动框体可缩小。在 ***BoxZoomOutFactor*** 中可设置缩小系数。缩小时在框的前面会显示有一个十字。设置 ***BoxZoomingOutCrossVisible = False*** 可以将其禁用掉。

7.19.3 ZoomPadding

ZoomPadding 属性定义了在缩放操作后 3D 模型与图边距之间的空白空间量。***ZoomPadding*** 在移动图表或手动缩放时不起作用，例如通过鼠标滚动。另外， ***ZoomPadding*** 不适用于基于矩形的缩放。

在 WinForms 中设置 ***ZoomPadding***:

```
chart.View3D.ZoomPanOptions.ZoomPadding = new Padding(10, 30, 10, 10);
```

在 Wpf 中设置 ***ZoomPadding***:

```
chart.View3D.ZoomPanOptions.ZoomPadding = new Thickness(10, 30, 10, 10);
```

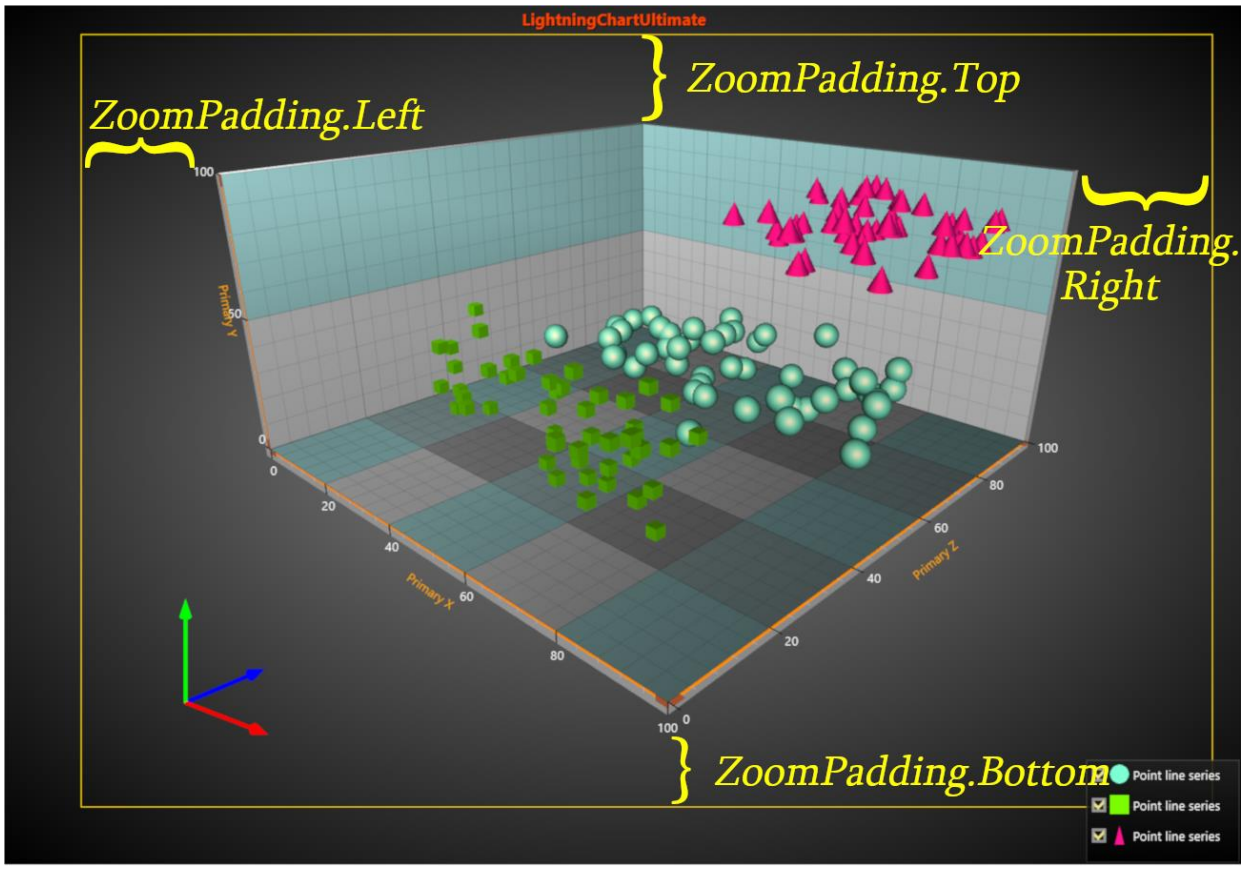


图 7-96. 若采用例如本例中的 ZoomToDataAndLabels 操作，ZoomPadding 会在数据/标签与图边距间留下空白空间。

7.19.4 ZoomToDataAndLabels

在 View3D 视图中， ***ZoomToDataAndLabels*** 操作使受图边距和 ***ZoomPadding*** 限制的可用区域，通过将摄像头移近/拉远来尽可能达到最佳效果。轴、标签、系列数据与标记都保持可见。图表标题、注释和图例框将因为它们的位置是在屏幕坐标中定义的而被忽略掉。用可以 ***ZoomToDataAndLabels*** 视图内容居中时，保持视场角。

默认情况下， ***LeftDoubleClickAction*** 属性设置为 ***ZoomToDataAndLabels***，此时双击鼠标左键可激活该操作。通过将该属性设置为 ***Off*** 可禁用。 在代码中，用 ***View3D.ZoomToFit（ZoomArea3D.DataAndLabelsArea）***方法可以调用 ***ZoomToDataAndLabels***。

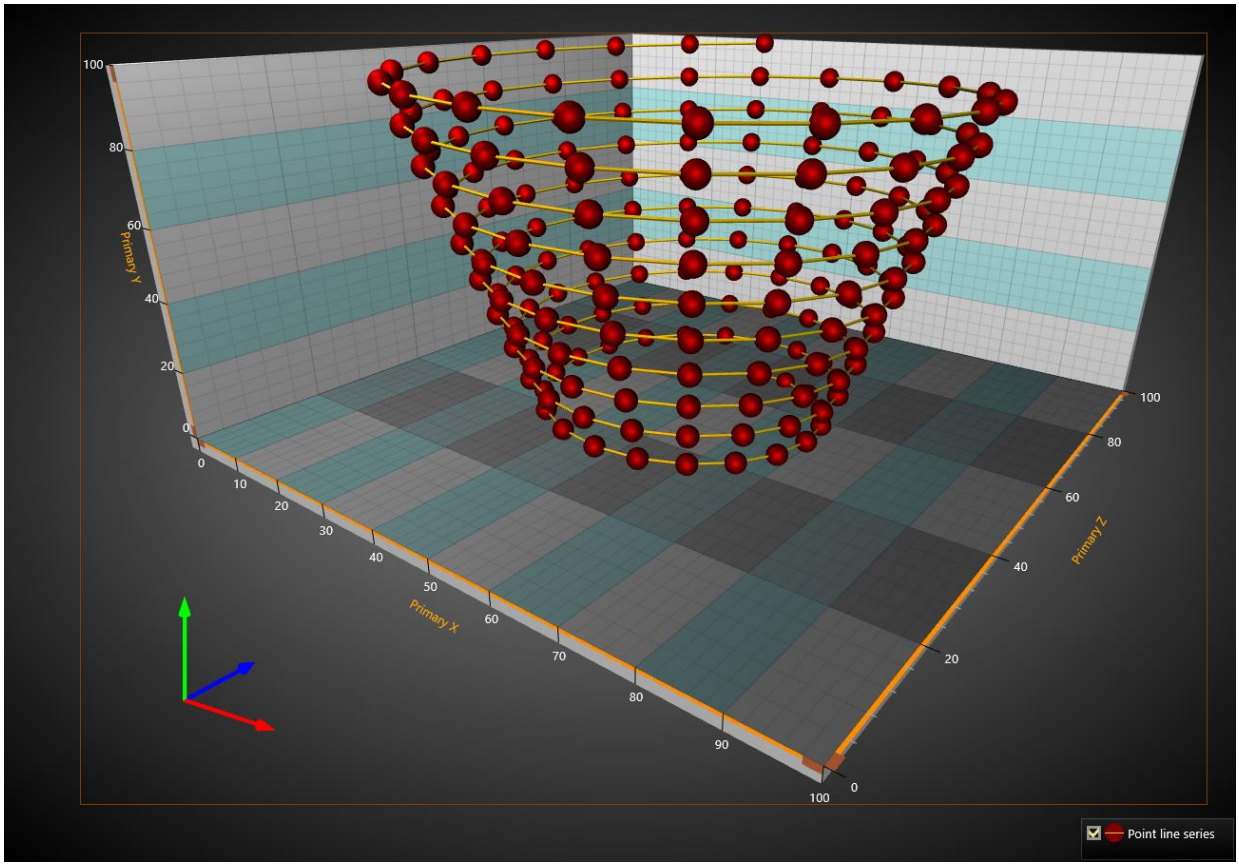


图 7-97. ZoomToDataAndLabels 操作已开启。数据系列、轴、墙等已在图边距内进行了最佳拟合。方向箭头位于图形区域的左下角，但忽略图例框。所有的边缘设置 ZoomPadding = 0，因此，在图边距和 DataAndLabelsArea 之间不存在空白区。

请注意，***ZoomToDataAndLabels*** 使用的是摄像头的 ***MinimumViewDistance*** 属性，这意味着在某些情况下，可能不会使用完整的可用图形区域，因为摄像头无法足够接近图表。本例中会发送一个 ***ChartMessage*** 通知。

7.19.5 旋转与平移

摄像头可以围绕 3D 模型旋转，只需按下鼠标按钮，并通过水平或垂直拖动。***RotationX***, ***RotationY*** 和 ***RotationZ*** 属性会更新。

当一个鼠标操作设置为 ***Pan***，平移时会更新 ***Camera*** 的 ***Target*** 属性。当鼠标操作设置为 ***PanPrimaryXZ***、***PanPrimaryXY*** 或 ***PanPrimaryYZ***，主 X 轴、Y 轴和 Z 轴范围会调整。例如，当用鼠标拖动， PanPrimaryXZ 会调整 X 和 Z 轴。副 X, Y 和 Z 轴不会改变。

设置 ***LeftMouseButtonAction*** / ***MiddleMouseButtonAction*** / ***RightMouseButtonAction*** 为 ***Pan/PanPrimaryXZ/PanPrimaryXY/PanPrimaryZ*** 以开启平移。 设置为 ***Rotate*** 可开启选择。若要禁用从鼠标左键平移和旋转，可将其设置为 ***None***。

用 ***PanSensitivity*** 可以控制应用的平移量。另外，用 ***RotationSensitivity*** 可控制应用的旋转量。

7.19.6 触摸屏进行缩放

两个手指放在图表上，将两指捏近可以缩小，展开可以放大。要禁用通过触摸屏来来缩放，可以设置 ***MultiTouchZoomEnabled*** 为 ***False***。

7.19.7 用触控屏幕平移

两个手指放在图表上，同时往同样的方向移动可进行平移。要禁用通过触控屏幕来平移，可以设置 ***MultiTouchPanEnabled*** 为 ***False***。

7.19.8 在轴上方用鼠标滚轮

当在轴上滚动鼠标滚轮，图表会进行特定轴的缩放或平移。通过 ***WheelAreaThickness*** 可以调整鼠标滚轮在轴附近的敏感区域宽度。用 ***AxisMouseWheelAction*** 可以在缩放与平移中进行选择。

7.19.9 通过代码缩放、选择与平移

通过 ***RotationX***, ***RotationY*** 和 ***RotationZ*** 属性来移动 ***View3D.Camera***，可以旋转 3D 视图。 当设定了 ***Perspective*** 和 ***Orthographic*** 摄像头后，通过设置 ***ViewDistance*** 可以完成缩放。设定 ***OrthographicLegacy*** 摄像头，通过变更 ***Dimensions*** 可完成缩放。通过设置摄像头的 ***Target*** 为 3D 模型坐标可以完成平移。

7.20 图例框

在 View3D 视图中的图例框与 ViewXY 视图中的图例框（参阅第 **Error! Reference source not found.**章节）非常的相似 。但是每个图形中仅允许使用一个图例框。而且，由于 View3D 中的轴不能分段，所以不存在基于分段的属性。通过 ***View3D.LegendBox*** 可以修改图例框属性。

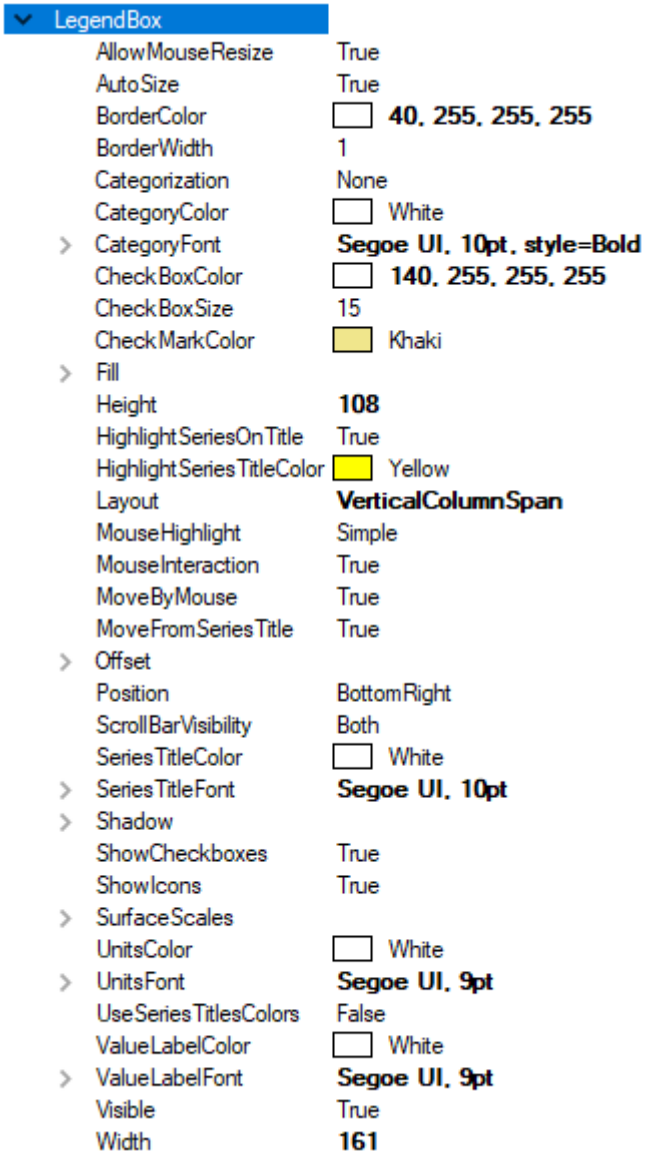


图 7-98. View3D 视图中的图例框属性

7.20.1 隐藏曲面系列调色板刻度

在 View3D 视图中代替 ViewXY 视图中 ***IntensityScales*** 属性的是 ***SurfaceScales*** 属性。要隐藏中的调色板刻度，可以设置 ***SurfaceScales.Visible = False***。要调整其大小，可以对 ***ScaleSizeDim1*** 和 ***ScaleSizeDim2*** 属性进行设置。

7.20.2 在 View3D 视图中对图例框进行定位

与在 ViewXY 视图中一样， View3D 中的图例框可以自动或手动布置。自动布置可以将其对齐到视图或者图形区域的左侧/顶部/右侧/底部。用 ***Position*** 属性可以对位置进行控制。一些定位选项会考虑到图边距，而也有一些并不是。

无视图边距的选项（将图例框置于图边距区域内）：

TopCenter*, *TopLeft*, *TopRight*, *LeftCenter*, *RightCenter*, *BottomLeft*, *BottomCenter*, *BottomRight*, *Manual

将图例框置于图边距区域内部的选项：

GraphTopCenter*, *GraphTopLeft*, *GraphTopRight*, *GraphLeftCenter*, *GraphRightCenter*, *GraphBottomLeft*, *GraphBottomCenter*, *GraphBottomRight

Offset 属性可以设置根据给定量从由 ***Position*** 属性确定的位置进行位置移动。

```
// 设置图例框位置，偏移量是从RightCenter位置移动
chart.View3D.LegendBox.Position = LegendBoxPosition.RightCenter;
chart.View3D.LegendBox.Offset = new PointIntXY（-15, -70）;
```

Manual 定位方法可以计算从图例框的左上角到视图左上角的偏移量。注意这与 ***TopLeft*** 选项不同， ***TopLeft*** 是从图形区域的顶部开始计算。

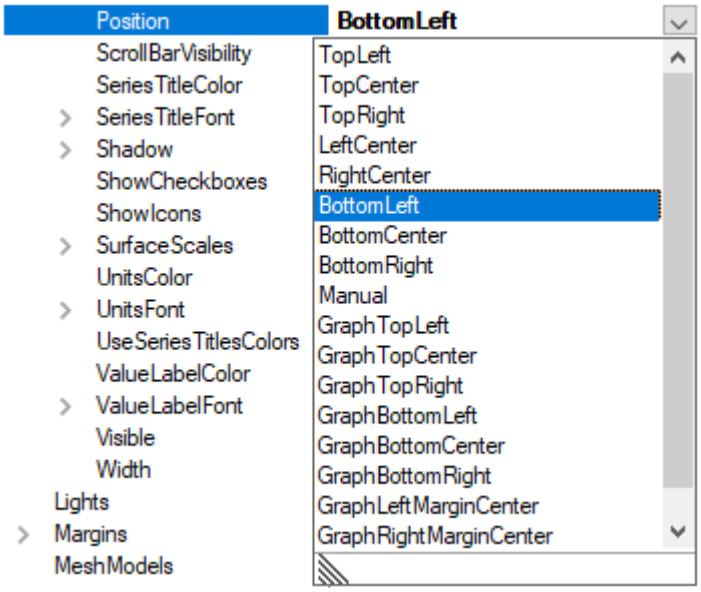


图 7-99. 图例框的定位选项；设定 Graph.. 类选项可将图例框放置于图边距内。

7.21 裁剪对象，保留轴值域内部分

演示示例： Simple 3D surface grid

通过设置 ***ClipContents*** 属性为 ***True***，系列、矩形和网眼模型均将裁剪以保留轴值范围内的部分。轴总是在一个维度上拉伸，因此当启用裁剪时，会阻止在墙外渲染。

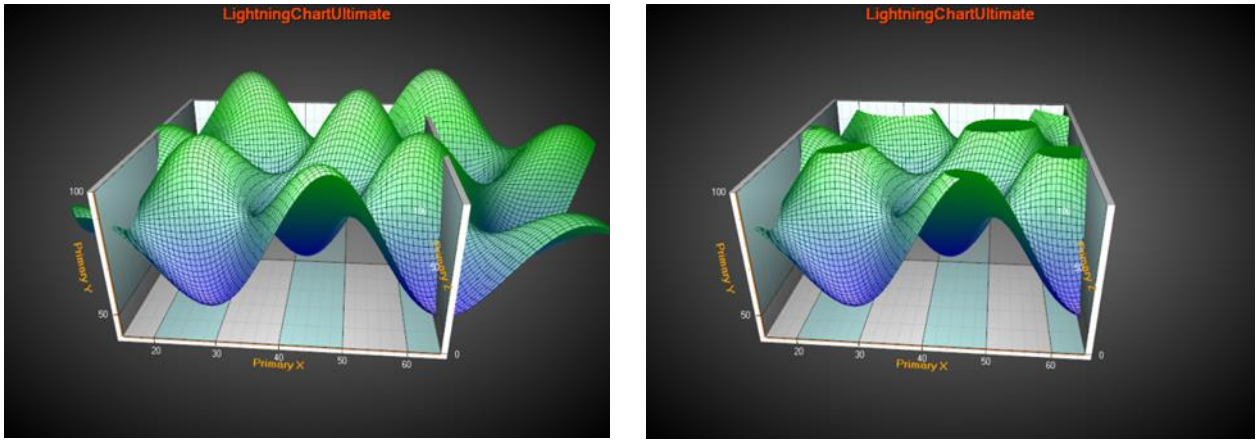


图 7-100. 左侧未使用 ClipContents；系列渲染轴值域之外的部分。右侧启用了 ClipContents

注意裁剪操作并不会改变系列数据集本身。裁剪只会在渲染阶段发生。而且，鼠标点击测试也会对墙外不可见的、剪切掉的对象起作用。

在启用裁剪之后，图表内所有的线条的宽度都会自动设置为 1.。

7.22 Annotation3D

演示示例： Points tracking; Surface mouse control; Mesh models coloring, wireframe

Annotation3D 集可以在 3D 场景中添加注释。通常，这与 ViewXY 中的 ***Annotations*** 类似（参阅第 6.26 章节）, 但使用 X、Y 和 Z 维度的 ***Target*** 和 ***Location*** 属性除外。

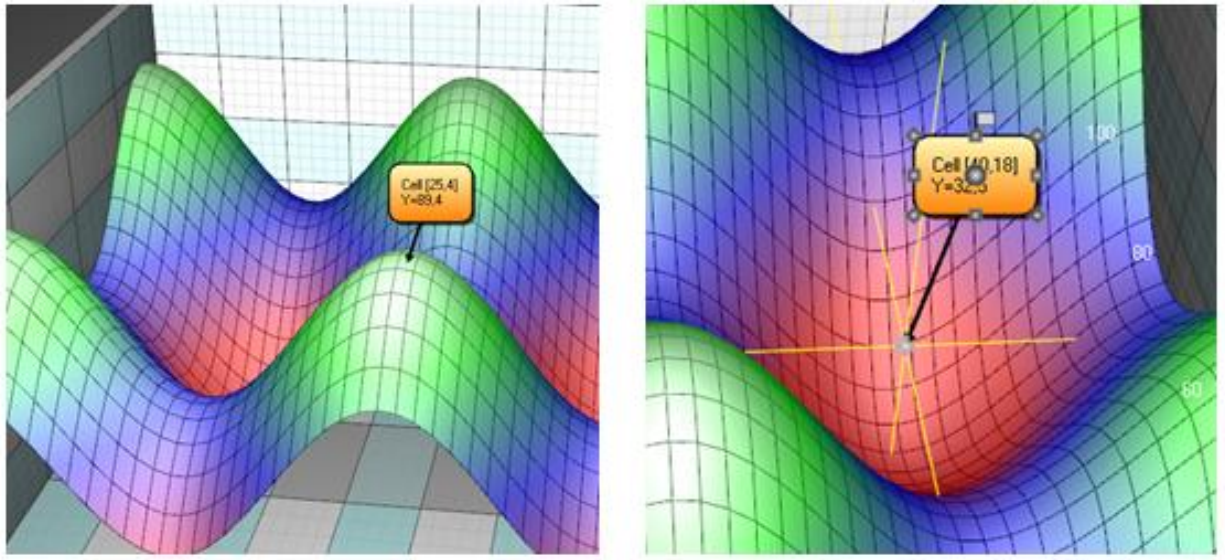


图 7-101. 显示一个 3D 系列值得 Annotation3D 对象。用十字光标可以辅助目标移动。

在 3D 视图中可以用鼠标移动 ***Target*** 。为方便移动，当鼠标移至 ***Target*** 节点上方时，注释会显示十字光标线。设置 ***ShowTargetCrosshair*** 属性为 ***Auto/On/Off*** ，并调整 ***TargetCrosshairLineStyle*** 中的线条样式。

8. 坐标系统转换器

以下坐标系统转换器可在 ***CoordinateConverters*** 名称空间中使用，该名称空间补充了 View3D 用途。

- Cartesian 3D <-> Spherical 3D
- Cartesian 3D <-> Cylindrical 3D

8.1 SphericalCartesian3D

演示示例： Spherical coordinates

SphericalCartesian3D 转换器类（converter class）可以在球面和 3D 笛卡尔坐标之间转换。

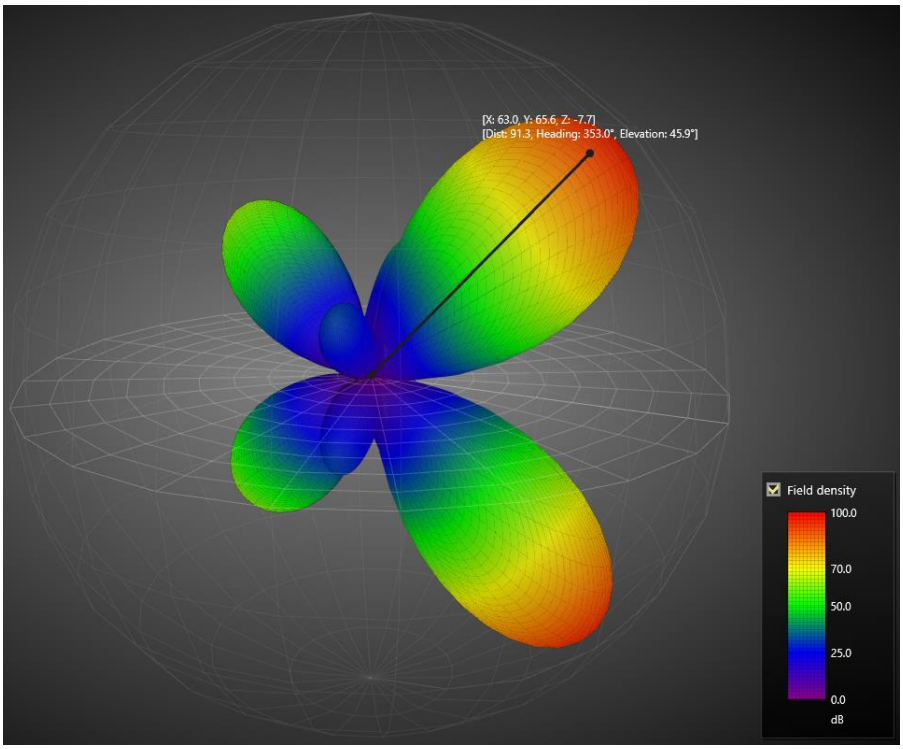


图 8-1. 用 SphericalCartesian3D 转换器创建的示例。用球形坐标定义 SurfaceMeshSeries3D 数据点与网格。 注释跟踪最近的数据点并以球形坐标显示其值。

球形数据点通过 ***SphericalPoint*** 对象来定义，这包括以下几个字段：

- ***Distance***: 与原点的距离 （0,0,0）
- ***ElevationAngle***: 高度角；也称作仰角或海拔，从 XZ 平面开始测量。ElevationAngle 是 90 度-倾斜角。
- ***HeadingAngle***: 航向角。又称方位角和绝对方位。

对仰角而言，XZ 平面是参考平面（例如赤道面）。仰角即为从该平面开始侧脸的角度。

注意! 这个转换器类认为 View3D.Dimensions 是均等的（立方体），否则转换结果可能必须通过用户端代码来标度。

View3D 的系列通常将数据以 X、Y、Z 值输入。这些值可以在例如 ***SeriesPoint3D***、***SurfacePoint3D*** 和 ***PointDouble3D*** 等对象中找到。

8.1.1 从球形转换为笛卡尔坐标

要将一个 ***SphericalPoint*** 转换为笛卡尔坐标，可以使用 ***SphericalCartesian3D.ToCartesian（）*** 方法。它接受以下数据输入

- SphericalPoint 点
- SphericalPoint[] 数组
- SphericalPoint[,] 矩阵

或者，使用 ***ToCartesian（）*** 扩展方法对球形点进行转换。

```
// 创建球形点矩阵
SphericalPoint[,] sphericalData = CreateSurfaceData（）;

// 将矩阵转换为笛卡尔坐标矩阵
SurfacePoint[,] xyzData = sphericalData.ToCartesian（）;
```

在 Bindable WPF 图表中将矩阵转换为笛卡尔坐标矩阵:

```
SurfacePointMatrix xyzData = sphericalData.ToCartesian（）;
```

8.1.2 从笛卡尔转换为球形

要将笛卡尔点转换为球形点，可以使用 ***SphericalCartesian3D.ToSpherical（）*** 方法。它接受采用 X、Y 和 Z 字段的 ***PointDouble3D*** 点作为数据输入。

或者，使用 ***ToCartesian（）*** 扩展方法对点进行转换。

```
// 定义笛卡尔点
PointDouble3D point = new PointDouble3D（50, 20, 40）;

// 转换为球形点
SphericalPoint sp = point.ToSpherical（）;
```

8.2 CylindricalCartesian3D

演示示例： Cylindrical coordinates

用于在圆柱形与 3D 笛卡尔坐标之间进行转换的转换器类。

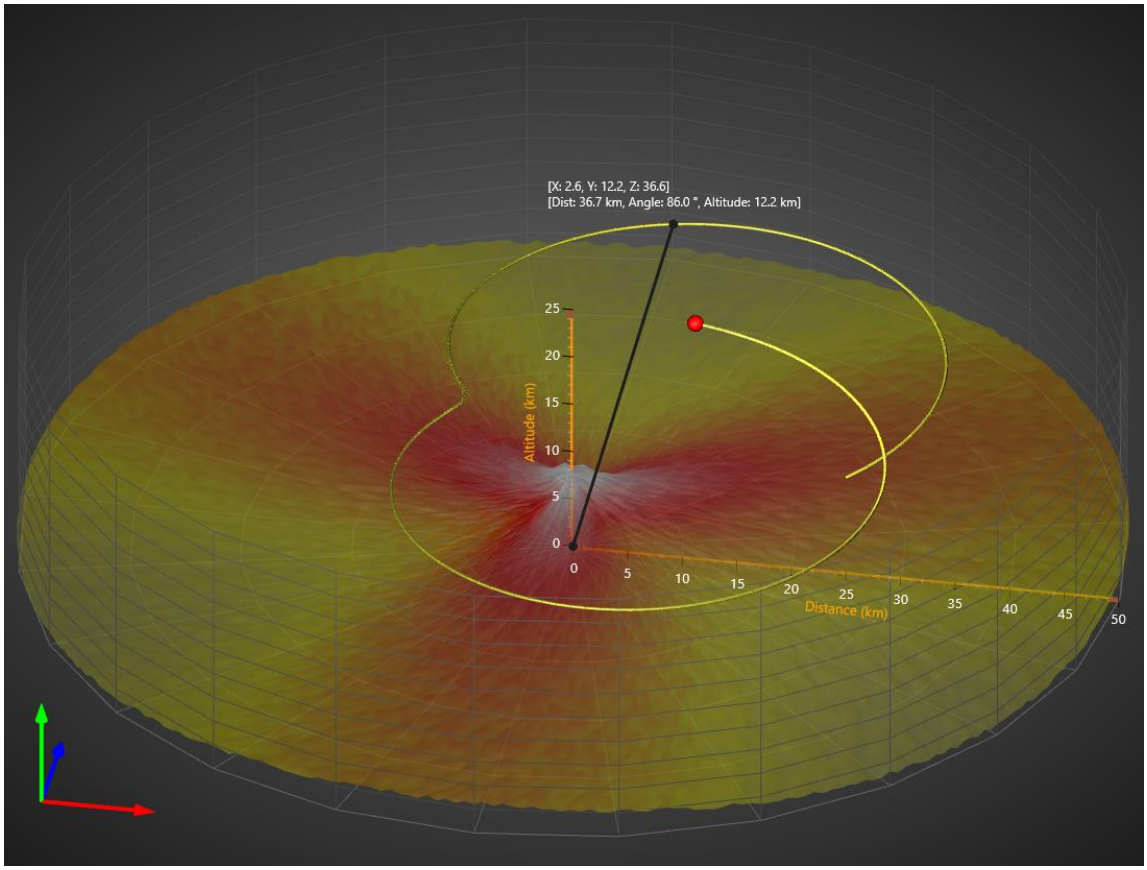


图 8-2. CylindricalCartesian3D 转换器创建的示例；用圆柱形坐标定义 SurfaceMeshSeries3D 数据点以及网格。 注释跟踪最近的 PointLineSeries3d 数据点并以圆柱形坐标显示其值。

圆柱形点由 ***CylindicalPoint*** 对象定义，包含以下字段：

- ***Distance***: 沿 XZ 平面的距离
- ***Y***: Y 值
- ***Angle***：航向角；又称方位角和绝对方位。

注意!这个转换器类认为 ***View3D.Dimensions.X*** 和 ***View3D.Dimensions.Z*** 是均等的，否则与 ***Angle*** 和 ***Distance*** （或者 X 与 Z）相关的转换结果可能必须通过用户端代码来标度。

View3D 的系列通常将数据以 X、Y、Z 值输入。这些值可以在例如 ***SeriesPoint3D***、***SurfacePoint3D*** 和 ***PointDouble3D*** 等对象中找到。

8.2.1 从圆柱形转换为笛卡尔

要将 ***CylindricalPoint*** 转换为笛卡尔坐标，可使用 ***CylindricalCartesian3D.ToCartesian*** （）方法。它接受以下数据输入

- CylindricalPoint 点
- CylindricalPoint[] 数组
- CylindricalPoint[,] 矩阵

或者，使用 ***ToCartesian（）*** 扩展方法对圆柱形点进行转换。

```
// 创建球形点矩阵
CylindricalPoint[,] cylindricalData = CreateData（）；

// 将矩阵转换为笛卡尔矩阵
SurfacePoint[,] xyzData = cylindricalData.ToCartesian（）；
```

在 Bindable WPF 图表中将矩阵转换为笛卡尔:

```
SurfacePointMatrix xyzData = cylindricalData.ToCartesian（）；
```

8.2.2 从笛卡尔转换为圆柱形

要将笛卡尔点转换为圆柱形点，可以使用 ***CylindricalCartesian3D.ToCylindrical（）*** 方法。它接受采用 X、Y 和 Z 字段的 ***PointDouble3D*** 点作为数据输入。

或者，使用 ***ToCartesian（）*** 扩展方法对点进行转换。

```
// 定义笛卡尔点
PointDouble3D point = new PointDouble3D（50, 20, 40）；

// 转换为球形点
CylindricalPoint sp = point.ToCylindrical（）；
```

9. ViewPie3D

演示示例： Pie 2D; Pie 3D; Donut 3D

ViewPie3D 以 3D 形式将数据呈现为饼图和圆环图。

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	

图 9-1. ViewPie3D 对象树

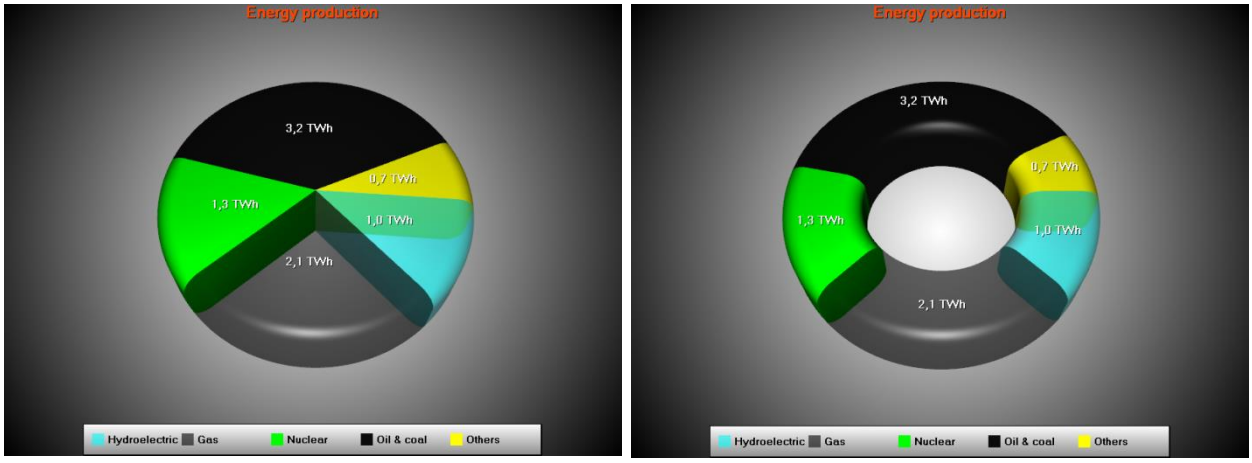


图 9-2. 3D 饼状图表与圆环图表示例

9.1 属性

用 ***Style*** 属性可以选择图表样式： ***Pie*** 或 ***Donut***。用 ***ZoomPanOptions*** 属性树可以管控缩放、平移与旋转效果，这与 View3D 类似（参阅第 **Error! Reference source not found.**章节）。

用 ***Camera*** 属性可控制视角（参阅第 **Error! Reference source not found.**章节）。用 ***LightingScheme*** 属性可以选择预定义光照设置。 用 ***Material*** 属性及其子属性可调整一般的 3D 表面外观和光泽度。

用 ***DonutInnerPercents*** 可设置圆环内径，用 ***Rounding*** 可以调整倒角内径，用 ***StartAngle*** 来旋转饼图，用 ***Thickness*** 可调整饼图厚度。当切片的 ***Explode*** 属性设置为 true 时，用 ***ExplodePercents*** 可以调整分解的扇形片相距多远。

用 ***TitlesStyle*** 可设置扇形片文本为以下之一： ***Titles***、***Values*** 或 ***Percents***。例如，编辑 ***TitlesNumberFormat*** 为“0.0 TWh”，可在最后包含单位。

Annotations 可以像在 View3D 中一样使用，但是没有轴值绑定属性（参阅第 7.22 章节）.

9.2 扇形片

饼图数据存储于 ***Values*** 集中。列表中的每一项都属于 ***PieSlice*** 类型。在 ***Value*** 属性中可以编辑数据值。在 ***Title.Text*** 属性中可设置标题字符串。通过定义 ***TitleAlignment = Outside***，可在饼图外绘制标题。

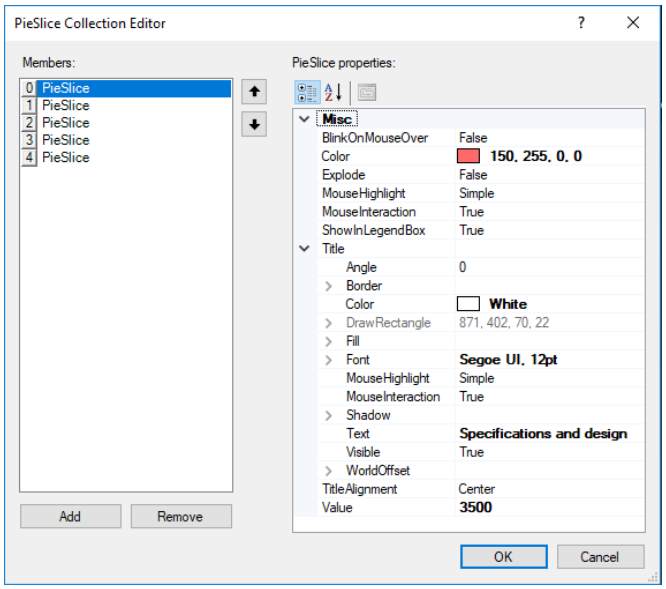


图 9-3. 饼图的 Values 列表编辑器

9.3 用代码设置数据

数据以 ***PieSlices*** 的形式存储在 ***Values*** 列表中。

```
//添加扇形片数据
//通过用true作为最后的参数，切片可自动添加至chart.ViewPie3D.Values集
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;
```

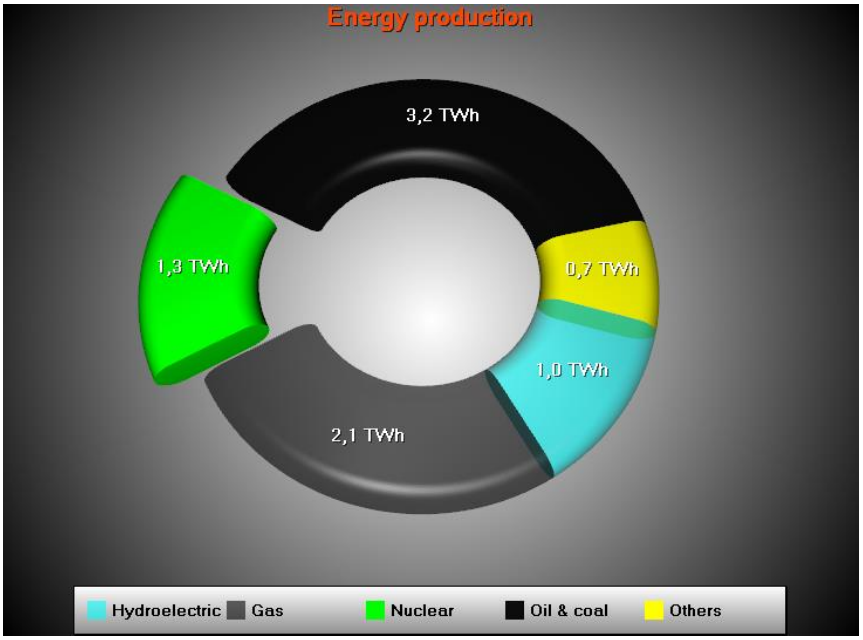


图 9-4. 数据集转成图表。通过采用 slice3.Explode = true 将第三个扇形片分开。

9.4 以 2D 方式查看饼状图

将摄像头设置为从顶部视角的预定义摄像头。

```
chart.ViewPie3D.Camera.SetPredefinedCamera（PredefinedCamera.PieTop）;
```

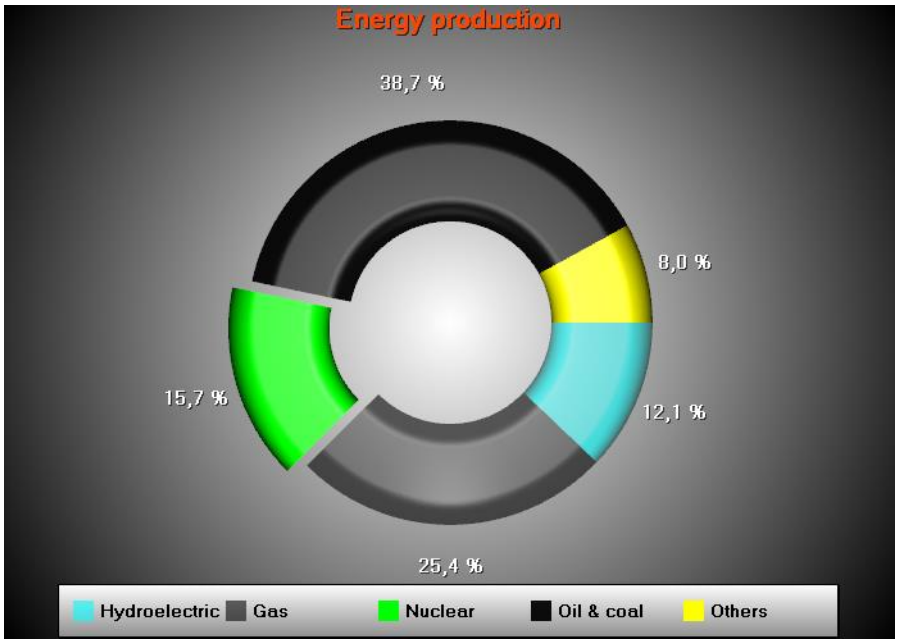


图 9-5. 采用顶部视角的预设摄像头的 2D 饼状图。

10. ViewPolar

ViewPolar 可实现以极面格式进行数据可视化。数据点位置由角值和振幅决定（比较在 ViewXY 中，角值相当于 X 和振幅相当于 Y）. Polar view 也具有缩放与平移功能。

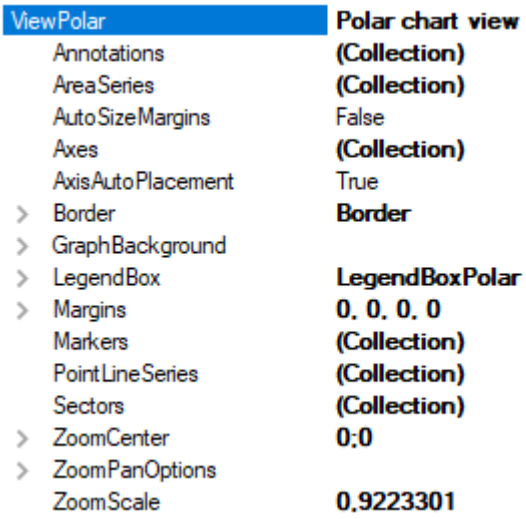


图 10-1. ViewPolar 对象树

10.1 Axes 轴

通过 ***Axes*** 属性列表可以定义极面轴。在同一个图表中可以使用多个轴。通过设置一个系列的 ***AssignPolarAxisIndex*** 属性，系列可以用这些轴中的任何一个来赋值。一个轴可以表示角度标度和振幅标度。另外，极面轴与 ViewXY 轴（参阅第 6.2 章节）非常相似。

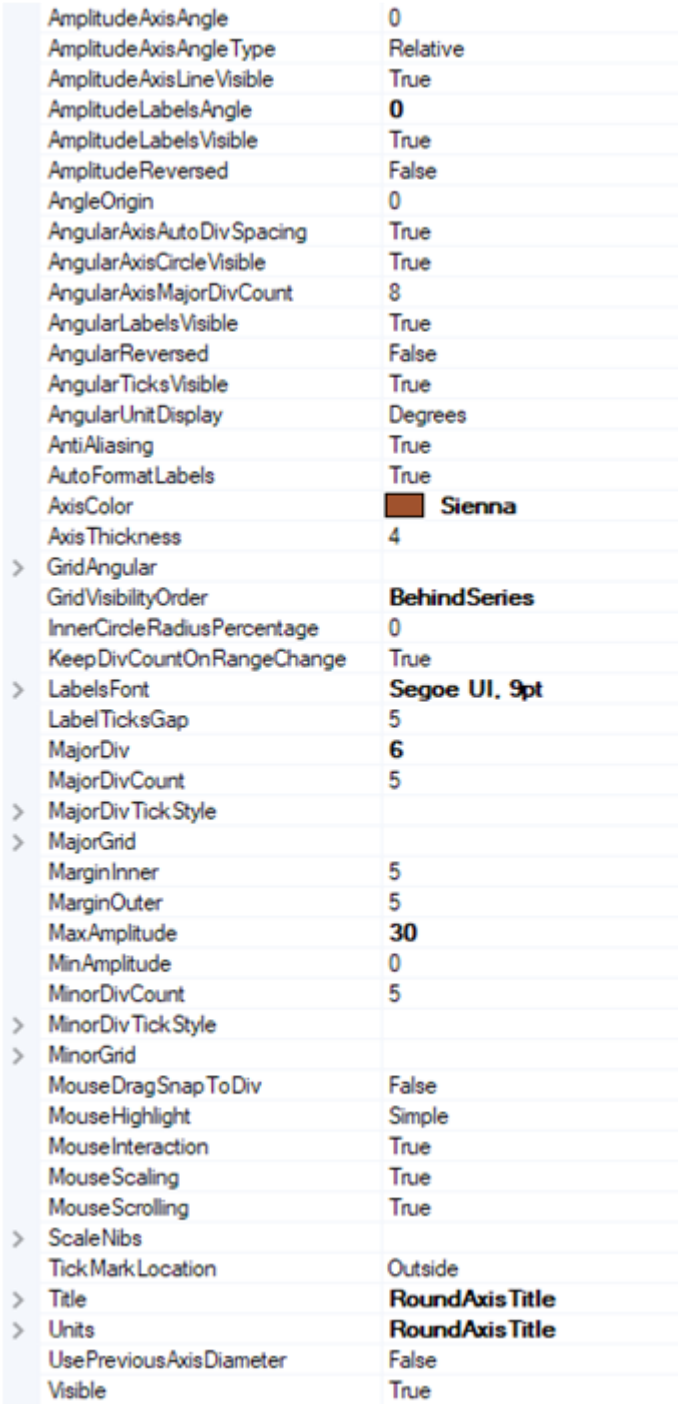


图 10-2. AxisPolar 属性树

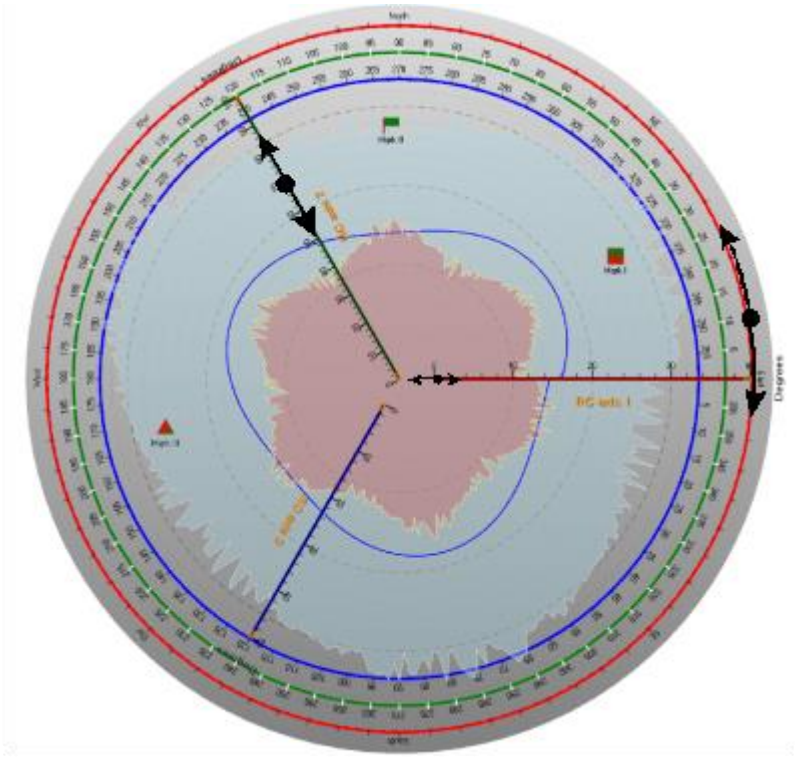


图 10-3.三个坐标轴中，第一个（红色）在外圈，第二个（绿色）在中间，第三个（蓝色）最靠近中心。 轴 AngleOrigin 可以通过将其拖动到轴圈上方来变更。振幅范围可以通过从轴拖动来改变。轴幅度范围的最小或最大值可以通过从振幅标度末端的小尖头进行拖动来变更。

10.1.1 逆向反转轴

轴可以通过振幅、角度或两者进行反转。要反转角度标度，可以设置 ***AngularReversed = True***。要反转振幅标度，可以设置 ***AmplitudeReversed = True***。

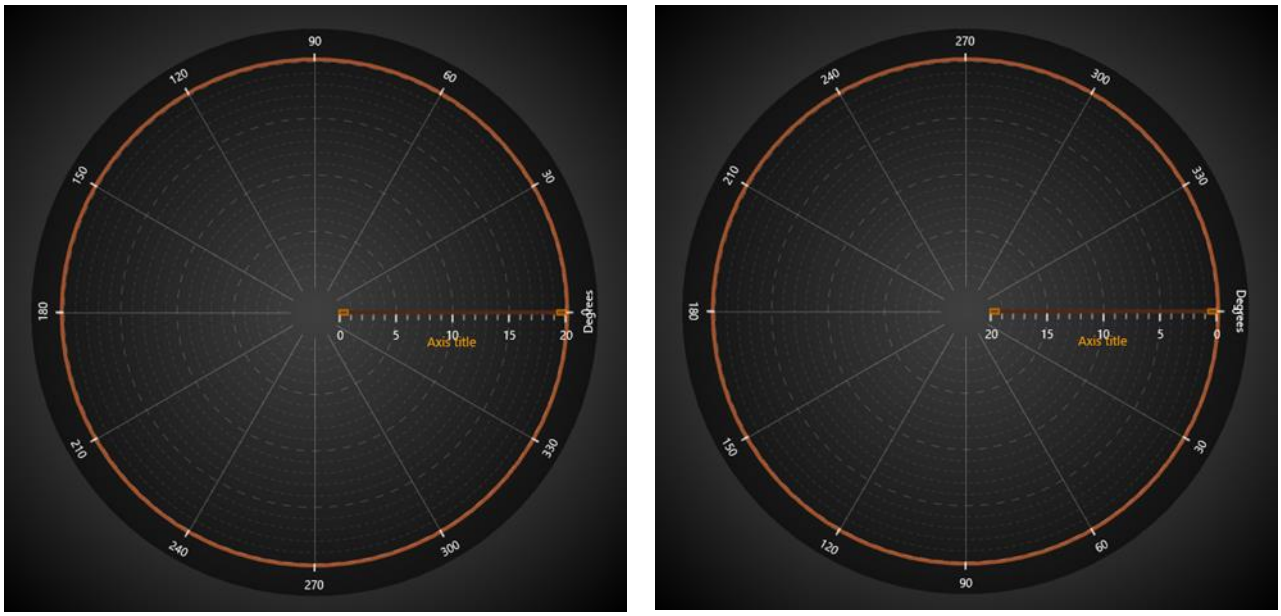


图 10-4. 左侧，标度没有反转。右侧，设置 AngularReversed = True 及 AmplitudeReversed = True.

10.1.2 设置标度的旋转角度

用 ***AngleOrigin*** 可以设置角度标度的旋转角度。

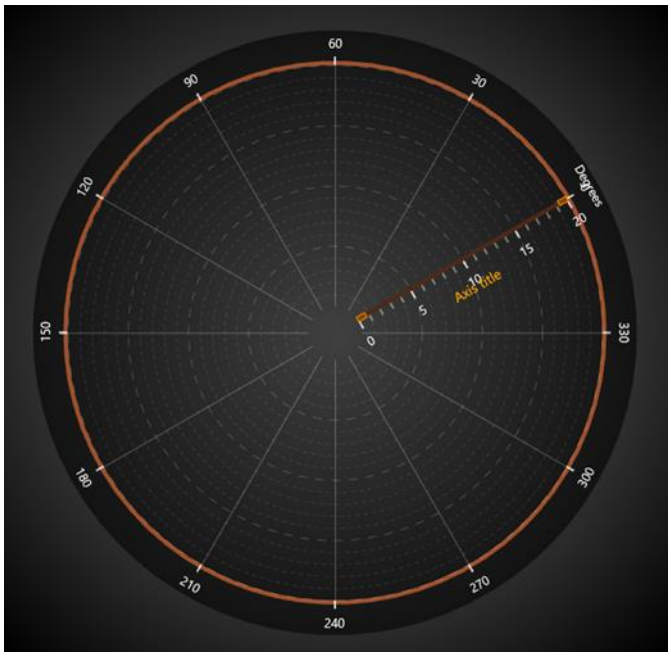
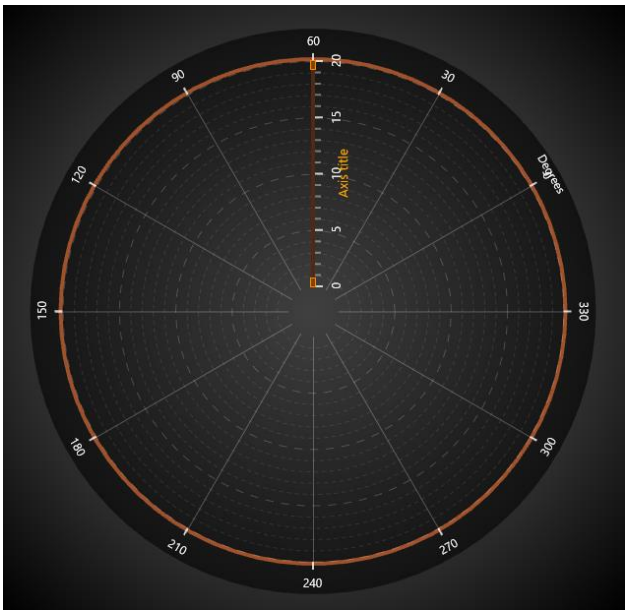
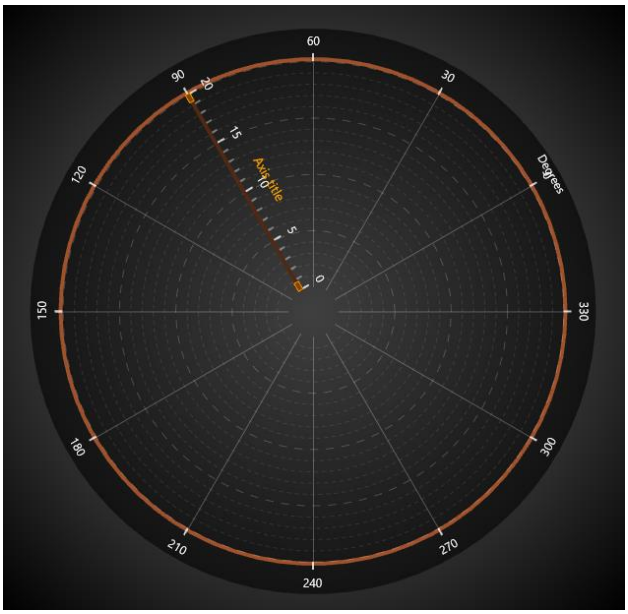


图 10-5. AngleOrigin = 30.

用 ***AmplitudeAxisAngle*** 可旋转振幅轴位置。振幅刻度角可以设置为绝对角度（***AmpitudeAxisAngleType = Absolute***），也可以设置为相对于角度标度的角度（





AmpitudeAxisAngleType = Relative）。

图 10-6. AngleOrigin = 30. AmplitudeAxisAngle = 90. 左侧，设置 AmplitudeAxisAngleType = Absolute；右侧，AmplitudeAxisAngleType = Relative；在本例中，振幅刻度总体旋转 120 度。

10.1.3 设置分格

用 ***MajorDivCount*** 可设置振幅分格量，用 ***MajorDiv*** 属性可设置分格大小。振幅标度将自动作相应调整（更新 ***MaxAmplitude***）。用 ***MinorDivCount*** 可设置振幅的最小分格量。

默认情况下，图表会尽量包含尽可能多的角度分格。要管控角度分格，可以设置 ***AngularAxisAutoDivSpacing*** 为 ***False***。然后图表会尝试分格的 ***AngularAxisMajorDivCount*** 量。如果图表空间太小，无法渲染所有的分格和标签，它将采用可适应的较少分格量。

10.2 Margins （图边距）

当设置 ***AutoAdjustMargins*** 为 ***enabled*** 后，图形大小会得到调整，以便有足够的空间放置所有轴和图表标题。当设置其为 ***disabled*** 后，应用 ***ViewPolar.Margins*** 属性，可以手动设置图边距。

在运行时，可以通过调用 ***ViewPolar.GetMarginsRect*** 方法以像素为单位检索图边距矩形，该方法既适用于自动图边距，也适用于手动图边距。当需要进行基于屏幕坐标的计算或对象布置时，这非常有用。

ViewPolar.MarginsChanged 事件可设置为在更改图边矩矩形（例如调整其大小）时触发。

视图的内容在图边距之外的部分会自动裁剪掉。除了图表标题、注释和图例框之外的其他所有内容都会被剪切掉，这是因为它们的位置是以屏幕坐标定义的，使得它们可以任意地放置在图边距上。 还可以绘制一个 1 像素宽的边框矩形 ***Border***,来显示图边距的位置。默认情况下，在 ViewPolar 视图中该边框不可见。此矩形的颜色可以通过 ***Border.Color*** 来变更。

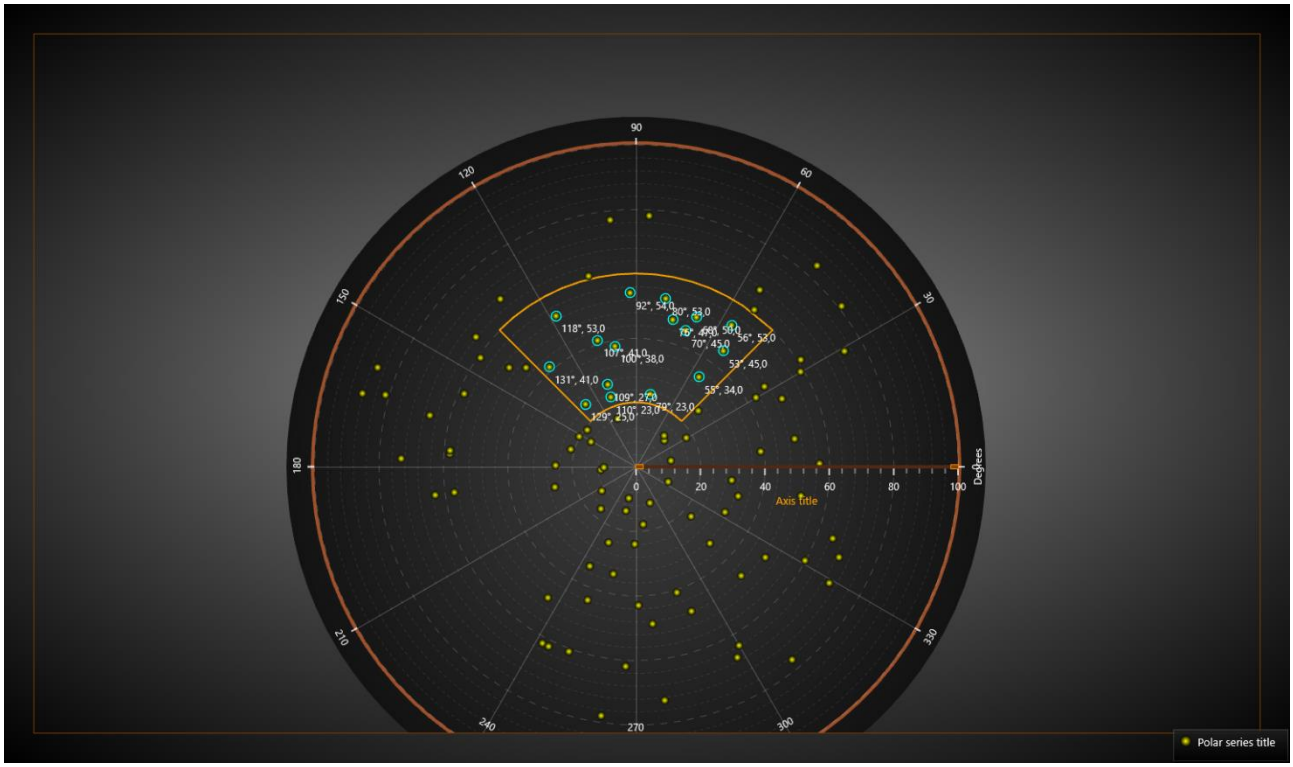


图 10-7. 极面图表的内容在图边距之外的部分会自动裁剪掉。绘制的边框可标记出图边距区域。轴标签在边框线内可见。

10.3 图例框

通过 ***ViewPolar.LegendBox*** 可以修改图例框属性。与 ViewXY 视图不一样的是, ViewPolar 视图只能设有一个图例框。

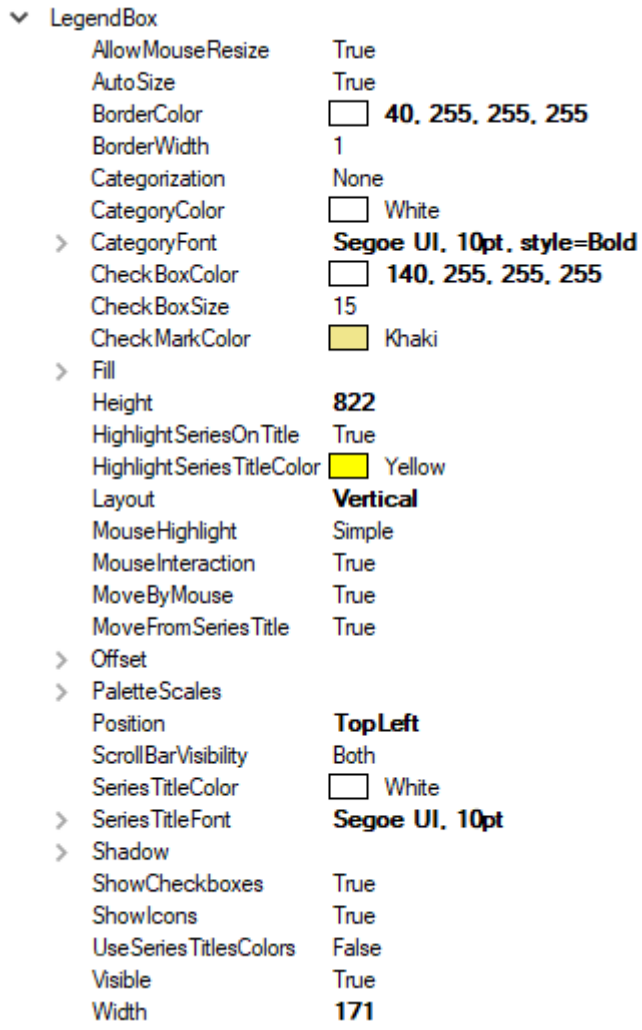


图 10-8. ViewPolar 中的图例框属性

10.3.1 Hiding palette scales 隐藏调色板标度

要隐藏一个图例框中调色板的标度，可以设置 ***PaletteScales.Visible = False***。要对齐进行调整，可以设置 ***ScaleSizeDim1*** 与 ***ScaleSizeDim2*** 属性。

10.3.2 在 ViewPolar 中定位图例框

ViewPolar 视图的图例框可以自动布置或可手动布置。自动布置可以将其对齐到视图或者图形区域的左侧/顶部/右侧/底部。用 ***Position*** 属性可以对位置进行控制。一些定位选项会考虑到图边距，而也有一些并不是。

无视图边距的选项（将图例框置于图边距区域内）：

TopCenter，***TopLeft***，***TopRight***，***LeftCenter***，***RightCenter***，***BottomLeft***，***BottomCenter***，***BottomRight***，***Manual***

将图例框置于图边距区域内部的选项：

GraphTopCenter, ***GraphTopLeft***, ***GraphTopRight***, ***GraphLeftCenter***, ***GraphRightCenter***, ***GraphBottomLeft***, ***GraphBottomCenter***, ***GraphBottomRight***

通过 ***Offset*** 属性可以设置根据给定量从由 ***Position*** 属性确定的位置进行位置移动。

```
// 设置图例框位置，偏移量是从RightCenter位置移动
chart.ViewPolar.LegendBox.Position = LegendBoxPosition.RightCenter;
chart.ViewPolar.LegendBox.Offset = new PointIntXY（-15, -70）;
```

用 ***Manual*** 定位方法可以计算从图例框的左上角到视图左上角的偏移量。注意这与 ***TopLeft*** 选项不同， ***TopLeft*** 是从图形区域的顶部开始计算。

10.4 PointLineSeriesPolar

演示示例： Line series, sector; Palette-colored line series; Event-colored line series; Scatter points selecting

ViewPolar 视图中的 ***PointLineSeries*** 系列可以用来绘制一条线、一组点或者一条点线。在 ***LineStyle*** 和 ***PointStyle*** 属性中，有许多线与点的样式可供选用。

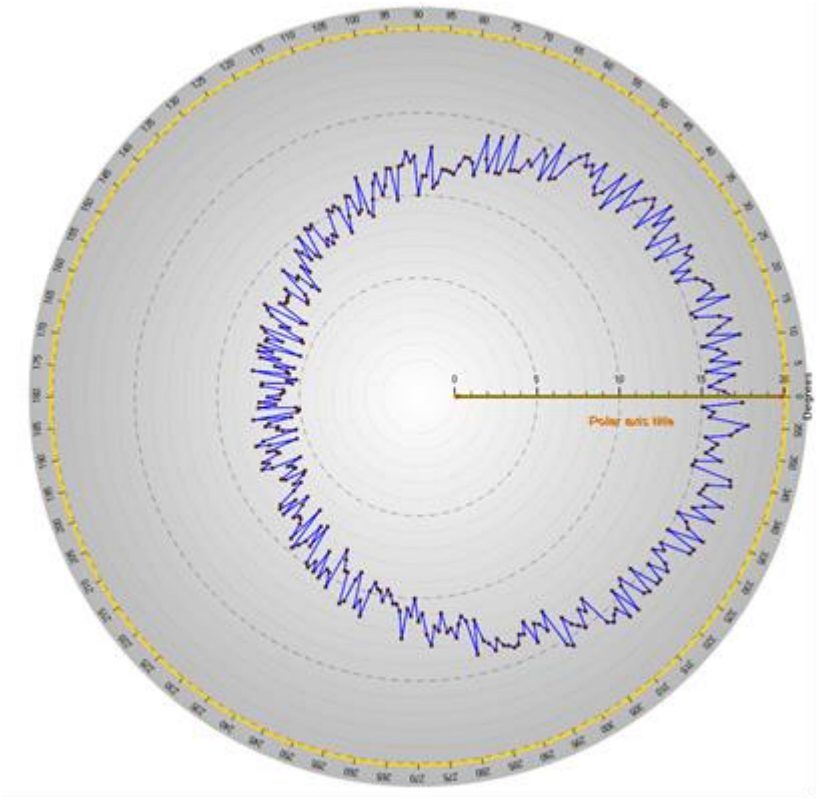


图 10-9. 用 ViewPolar 的 PointLineSeries 来呈现一些数据。线与点都可见。

10.4.1 设置数据

表示前一图中数据设置的代码如下。

```
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;
```

仔细查看前面的图像可以发现，第一个和最后一个数据点并没有连接起来。***PointLineSeriesPolar* 的 *ClosedLine*** *属性启用后，会动在这些点之间画一条线。*

```
//连接第一个和最后一个数据点。
pointLineSeriesPolar.ClosedLine = true;
```

10.4.2 调色板着色

线条支持调色板着色。利用 ***ColorStyle*** 属性可选择应用调色板着色的方式。

- ***LineStyle***: 无颜色填充。应用在 LineStyle.Color 属性中设置的颜色。
- ***PalettedByAngle***: 数据点的 ***Angle*** 字段决定颜色。
- ***PalettedByAmplitude***: 数据点的 ***Amplitude*** 字段决定颜色。
- ***PalettedByValue***: 数据点的 ***Value*** 字段决定颜色。

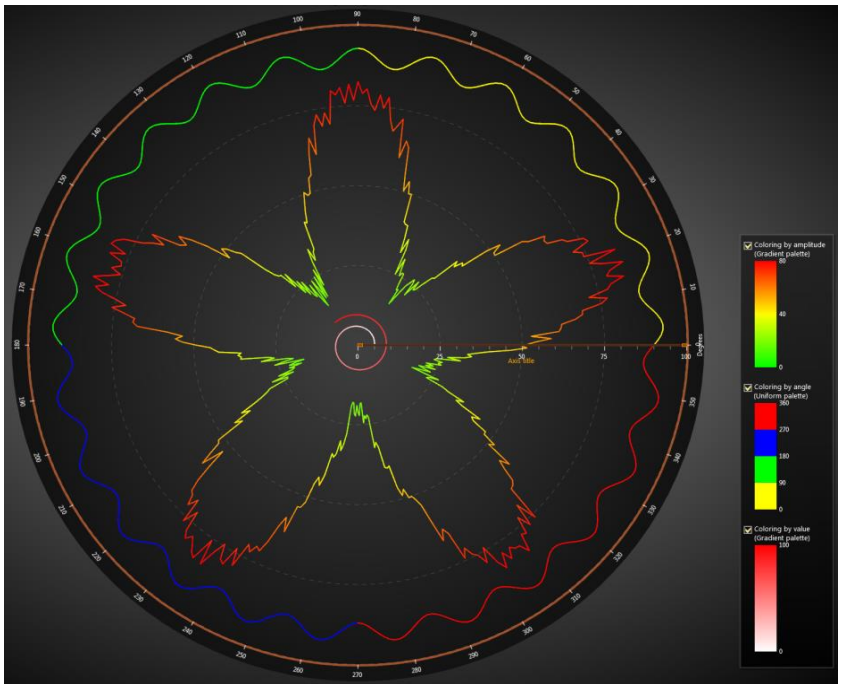


图 10-10. 应用的调色板着色

用 ***ValueRangePalette*** 属性可定义颜色与值阶，这种方法与 ViewXY 和 View3D 中的系列相似

10.4.3 用 CustomLinePointColoringAndShaping 事件自定义外形与着色

使用 ***CustomLinePointColoringAndShaping*** 事件可以进行自定义着色和坐标调整，该事件在进入图表的渲染阶段之前调用。其工作方式与 ViewXY 中 ***FreeformPointLineSeries*** 的 ***CustomLinePointColoringAndShaping*** 事件类似（参阅第 6.14.2 章节）。

10.5 AreaSeries

演示示例： Area series; Combined with markers; Spider / radar chart; Speedometer gauge

区域系列可以填充的区域样式实现数据可视化。在边缘的线条样式可以用 ***LineStyle*** 属性来编辑。用 ***FillColor*** 属性可以对填充进行变更。

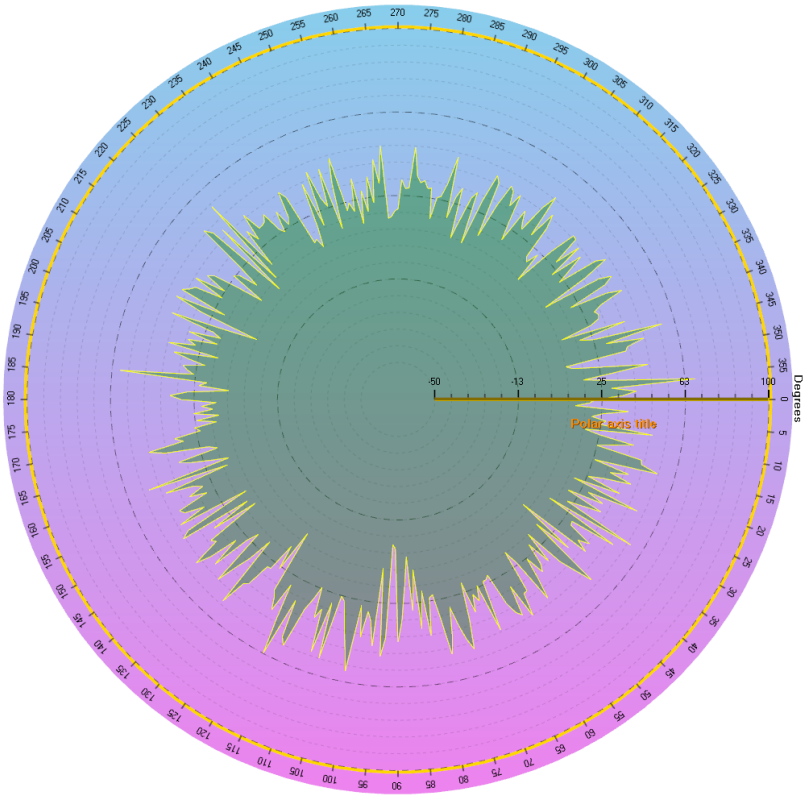


图 10-11. 用 ViewPolar 的 AreaSeries 展示一些数据。

10.5.1 设置数据

表示前一图中数据设置的代码如下。

```
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;
```

10.6 强度网格

IntensityGridSeriesPolar（极地热图）类似于 XY 图表中的强度网格系列。它允许可视化

M x N 节点数组，由指定的值范围调色板着色。节点之间的颜色是插值的。IntensityGridSeriesPolar 在极地空间中均匀分布。数据以双精度 [Angle, Amplitude] 的二维数组形式存储在 Data 属性中。MinimumAmplitude 和 MaximumAmplitude 属性定义应安装网格的幅度范围，而 BeginAngle 和 EndAngle 属性设置要安装的角度范围。对于流式/实时应用程序，用户应使用 Series.UpdateData() 方法。

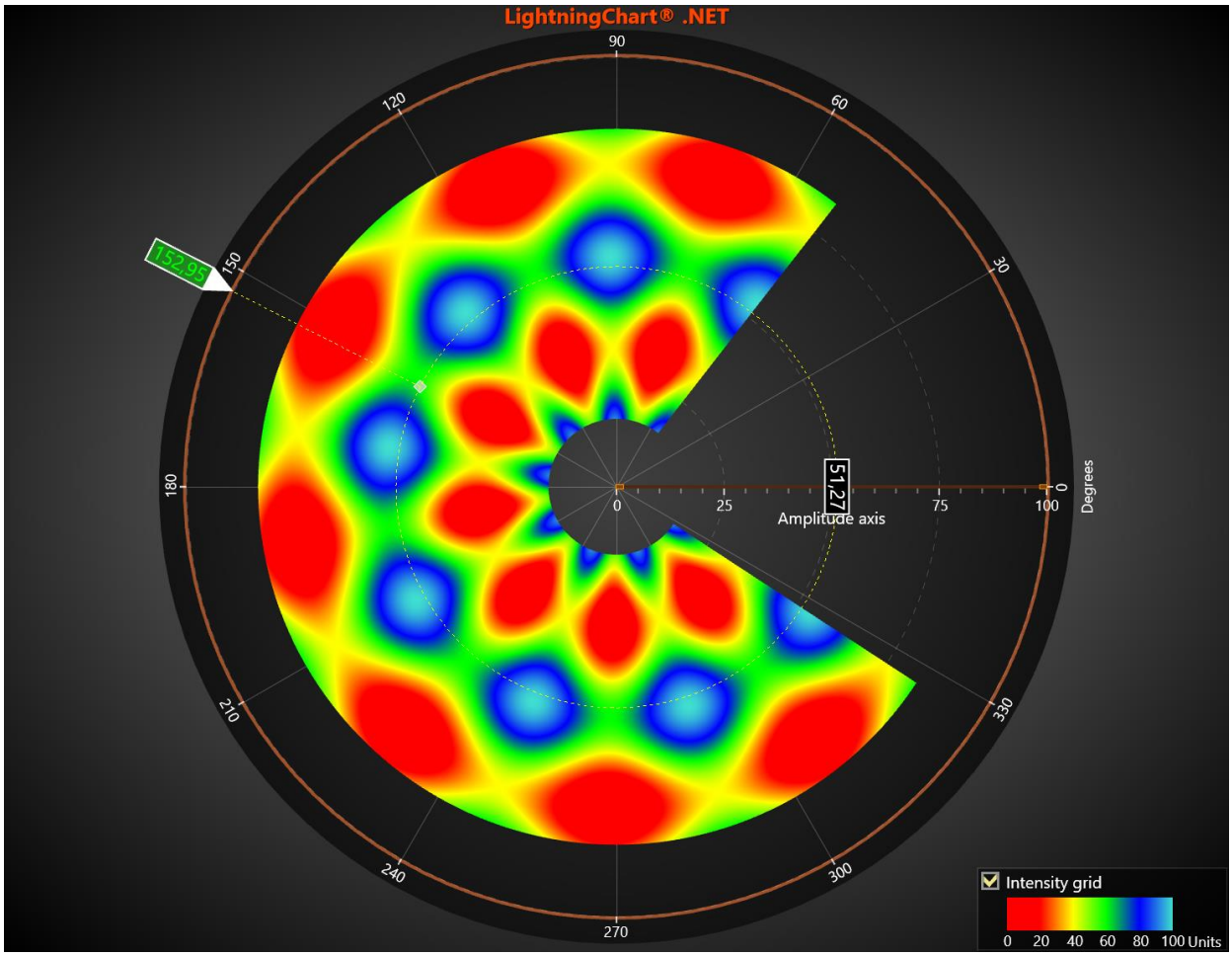


图 10 12. IntensityGridSeriesPolar。BeginAngle 和 EndAngle 用于定义角度范围

10.7 扇区

演示示例： Line series, sector; Wind rose diagram; Scatter points selecting; Scanning radar

Sectors 可以用来表示某个角度或振幅范围。用 ***MinAmplitude*** 和 ***MaxAmplitude*** 属性可以定义幅度范围。用 ***BeginAngle*** 和 ***EndAngle*** 可以定义角度范围。用鼠标拖动可以移动扇区。

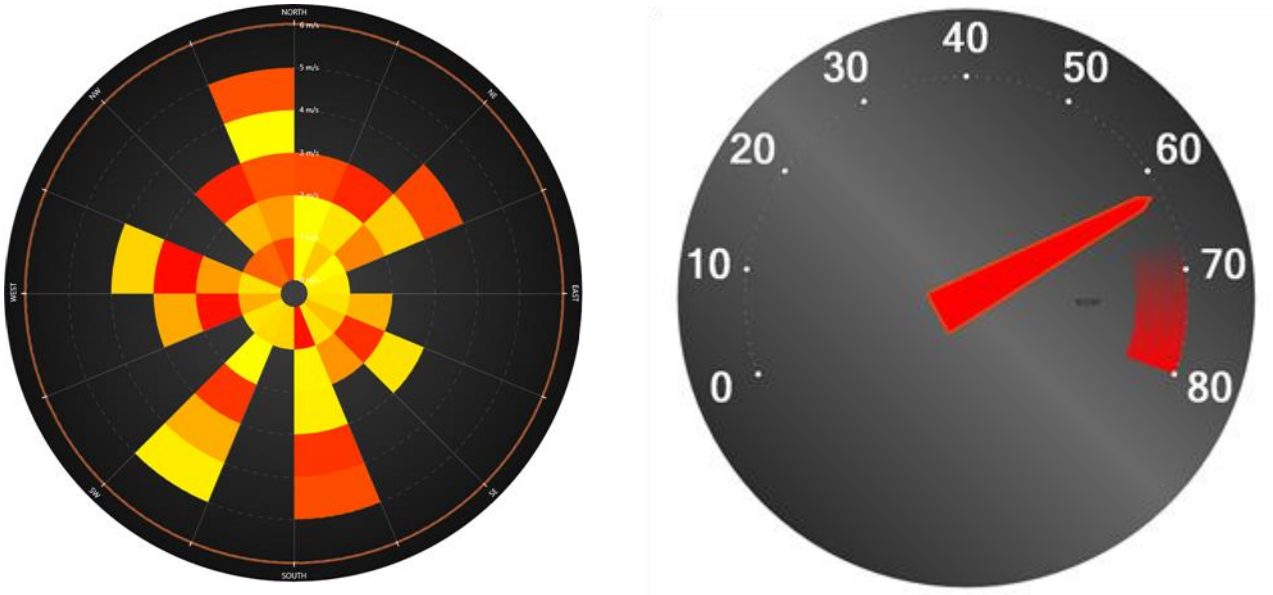


图 10-12. 采用扇区的两例。第一图为风玫瑰图（Wind Rose），由几个不同颜色的扇区生成。第二个图中， 用 AreaSeries 生成的一个刻度盘，用一个扇区表示转速表红色区域。

10.8 注释

演示示例： Vectors

Annotations 与 ViewXY 的 ***Annotations*** （参阅第 6.26 章节）类似，但以极面轴值定义的 ***Target*** 和 ***Location*** 除外。这里根据轴值调整大小并不适用，因此 ***Sizing*** 属性仅具有 ***Automatic*** 和 ***ScreenCoordinates*** 值。

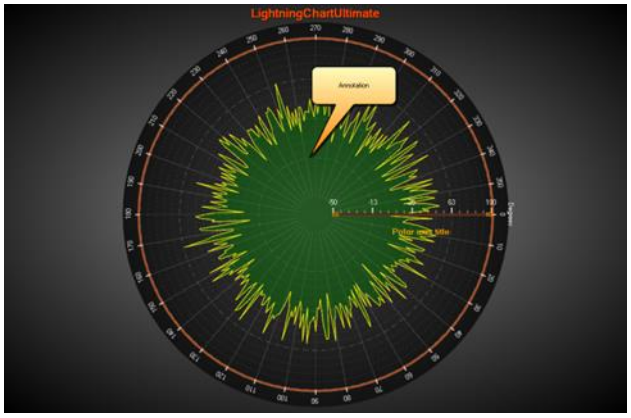


图 10-13. 一个极点视图中的注释

10.9 标记

Contents

10.10

演示示例： Scatter points selecting; Combined with markers; Sonar fish indicator; Scanning radar

标记符可以用来标记在某特定位置的特殊数据。通过设置标记的 ***AssignPolarAxisIndex*** 可以用首选轴分配标记。定义 ***Amplitude*** 和 ***AngleValue*** 属性可以放置标记。编辑 ***Symbol*** 可以设置首选外观，用 ***Label*** 属性可以定义标记文本。

用鼠标拖动可以移动标记。设置 ***SnapToClosestPoint*** 为 ***Selected*** 或 ***All***，可以在拖动时捕捉对齐最近的数据点。设置 ***Selected*** 则仅跟踪此标记被设置为用 ***SetSnapSeries***（）方法捕捉对齐的系列。设置为 ***All*** 可跟踪所有系列。

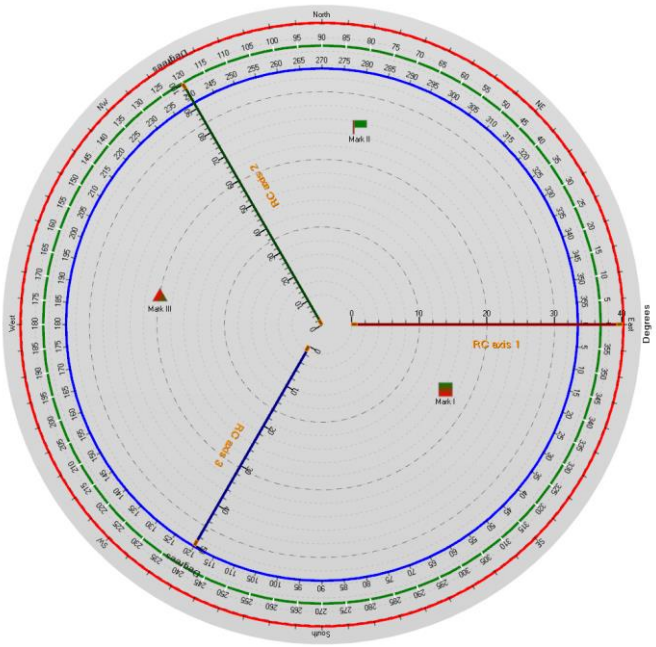


图 10-14. 在极面图表中的一些标记

10.11 数据游标

从 10.5 版本开始，ViewPolar 有一个内置的数据游标，它会自动跟踪最接近鼠标光标的系列值，并允许在结果表中显示它。 光标由振幅轴和角度轴的十字线、最近数据值位置的跟踪点、显示当前振幅和角度值的轴标签以及结果表组成，结果表格除了轴值外还显示系列名称和 它的颜色。

可以通过 Visible 属性启用或禁用数据游标。 也可以通过设置 ShowHaircrossLines 或 ShowLabels 或其他基于应该隐藏的内容的相应“显示”属性 false 来单独隐藏某些光标组件，例如线条或轴标签。 光标的外观也可以通过组件特定的属性进行修改。 LabelFont 修改轴标签文本，TrackingPointStyle 允许更改跟踪点。 Results 属性包含修改结果表的所有选项。 要修改单个组件，例如其中一个标签或线条，请使用配置属性下的选项。

```
// Enables data cursor but hides its axis labels.
_chart.ViewPolar.DataCursor.Visible = true;
_chart.ViewPolar.DataCursor.ShowLabels = false;

// Enabling and modifying the result table.
_chart.ViewPolar.DataCursor.ShowResultTable = true;
_chart.ViewPolar.DataCursor.Results.Background.Color = Colors.DarkBlue;
```

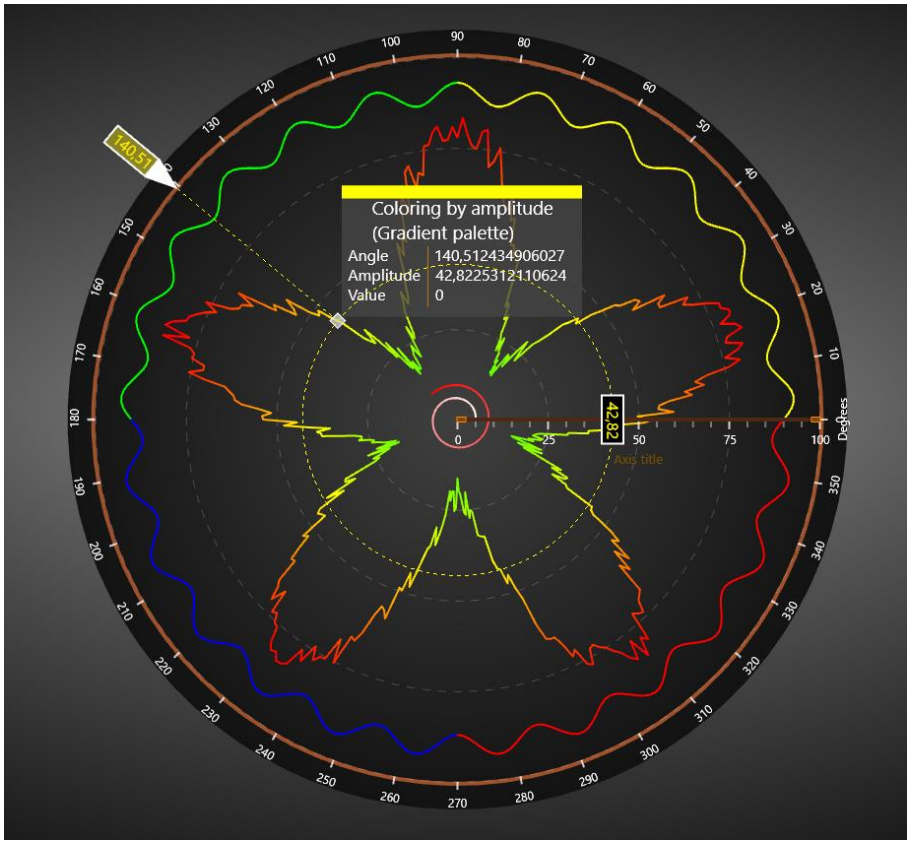


Figure 10-15. 极坐标视图中的数据光标。 结果表已通过启用 ShowResultTable 设置为可见。

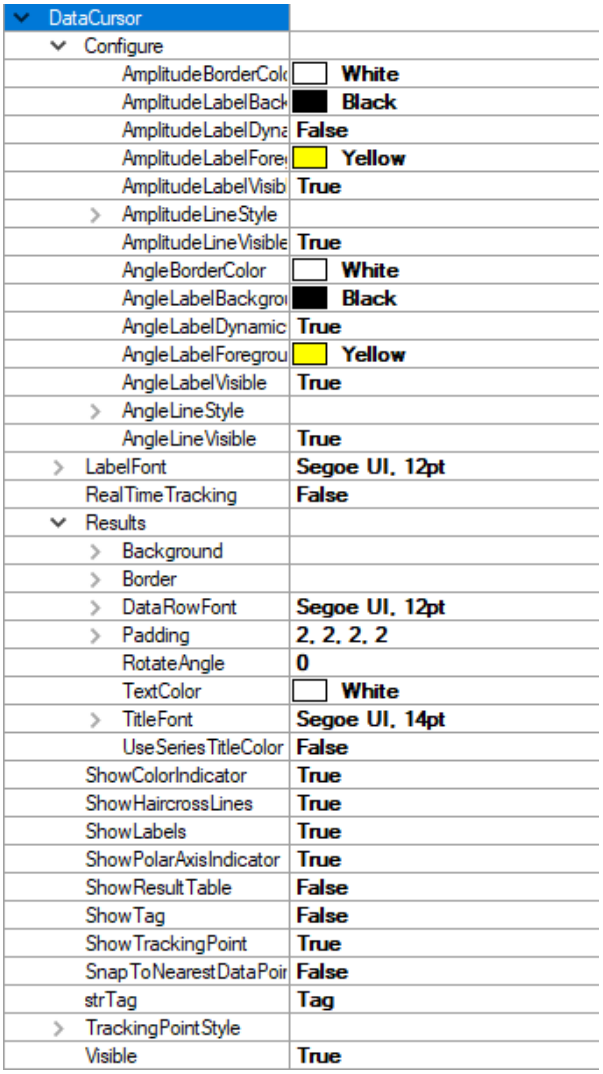


Figure 10-16. 数据游标的属性树。

10.12 缩放与平移

缩放可以通过代码设置 ***ZoomCenter*** 和 ***ZoomScale*** 属性来应用。 ***ZoomCenter*** 可定义为相对的 X-Y 范围。

X = -1：极面视图的左边缘在图表区域中心
X = 0：极面视图的中心在图表区域中心
X = 1：极面视图的右边缘在图表区域中心

Y = -1：极面视图的底边缘在图表区域中心
Y = 0：极面视图的中心在图表区域中心
Y = 1：极面视图的上边缘在图表区域中心

ZoomScale 为放大系数。例如，若值为 2，则使图表在 X 和 Y 方向上显示为 1 的两倍。

鼠标缩放功能可在 ***ZoomPanOptions*** 属性树种进行配置。

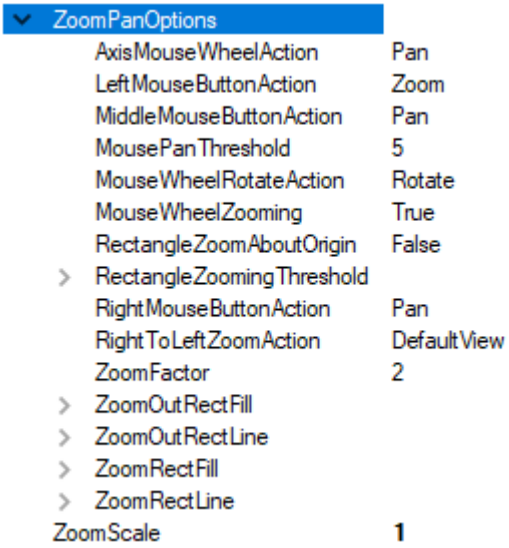


图 10-15. ViewPolar 的 ZoomPanOptions.

10.12.1缩放操作与方法

ZoomPanOptions 下的各种缩放操作可以设置为鼠标操作。***DefaultSettings*** 可返回初始缩放和置于中心位置设置。用 ***ZoomToData*** （在第 v8.4 版本前又称作 ***FitView***）可移动视点以显示图边距内的所有数据。***ZoomToLabelsArea*** 显示全部数据帧，包括图边距内的标签。

ViewPolar 具有 ***ZoomPadding*** 属性，与在（参阅第 7.19.3 章节）中的作用效果类似。.

缩放操作也能够以代码方式使用，如同用的 ***ZoomToFit（ZoomAreaRound.AreaName）***方法一样。例如调用 ***ZoomToFit（ZoomAreaRound.LabelsArea）***方法可以获得与通过鼠标动作进行 ***ZoomToLabelsArea*** 操作的一样结果。

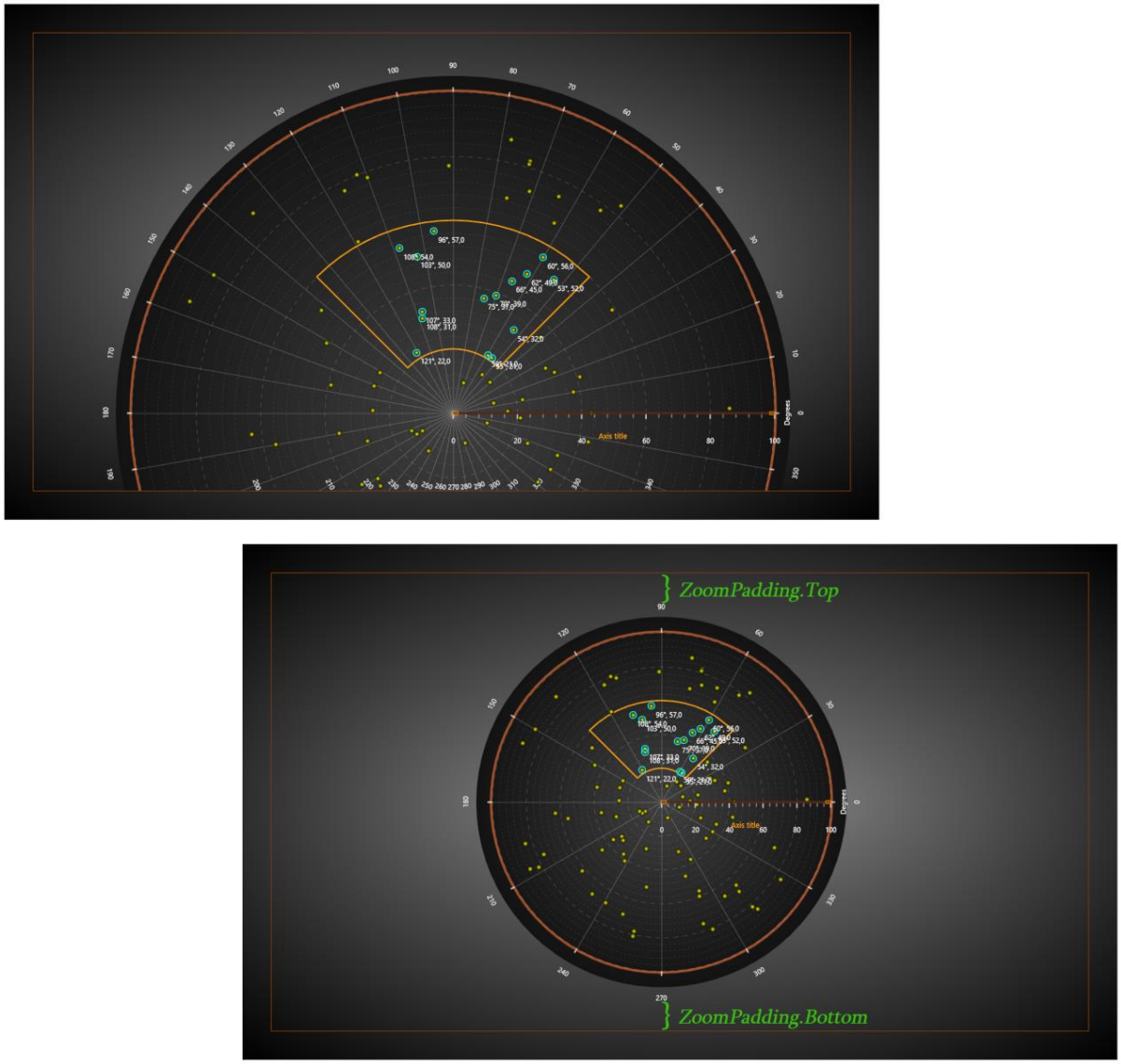


图 10-16. 在缩放操作（ZoomPadding = 50）前后的极面图表。在上方，图表已经过手动缩放，但没有调用缩放操作；而 ZoomPadding 没有起到作用。下方，采用 ZoomToLabelsArea, 在缩放时也会将标签考虑在内。

10.13 在 ViewPolar 中进行数据裁剪

PointLineSeriesPolar、AreaSeriesPolar、Sectors 和 PolarEventMarkers 具有 ***ClipInsideGraph*** -属性， 可以隐藏不在图形区域半径内的数据点。确切的裁剪点是最外层的角度轴。默认情况下，所有系列的 ***ClipInsideGraph*** 都为开启状态。

```
// 禁用图形外部裁剪
pointLineSeriesPolar.ClipInsideGraph = false;
```

在 LightningChart 版本 8.5.1 中引入了 ***CenterClipping***。它的工作原理与 ***ClipInsideGraph*** 类似，不过不同的是，比如当用鼠标拖动振幅轴时，控制在极面图表的中心的数据如何裁剪。***CenterClipping*** 具有以下三个选项可供选择：

-***None***: 在版本 8 之前就有。系列被移到中心点的另一侧，并且不会以任何方式裁剪（扇区除外）.

-***Center***:在图形中心点的数据被剪切，并且决不会被移动到相反的一边。此为默认选项。

-***InnerCircle***: 在振幅轴的最内层的数据被剪切，即轴的最小值或最大值，这具体取决于轴是否反转。如果图表中有多个振幅轴，则系列要根据其被分配的轴来裁剪。

```
// 当未采用反转轴时，设置PointLineSeriesPolar裁剪至振幅轴最大值以下
_chart.ViewPolar.Axes[0].AmplitudeReversed = false;
pointLineSeriesPolar.CenterClipping = CenterClipping.InnerCircle;
```

Center 和 ***InnerCircle*** -选项并不总是在同样的位置，因为 ***InnerCircleRadiusPercentage*** -属性可用于在图形中心附近留下空白区。换言之，它定义了一个振幅轴从何开始。
InnerCircleRadiusPercentage 特定于其所设置的轴，这意味着它不会影响其他振幅轴。

```
//为该轴将InnerCircleRadiusPercentage设置为10%
chart.ViewPolar.Axes[0].InnerCircleRadiusPercentage = 10;
```

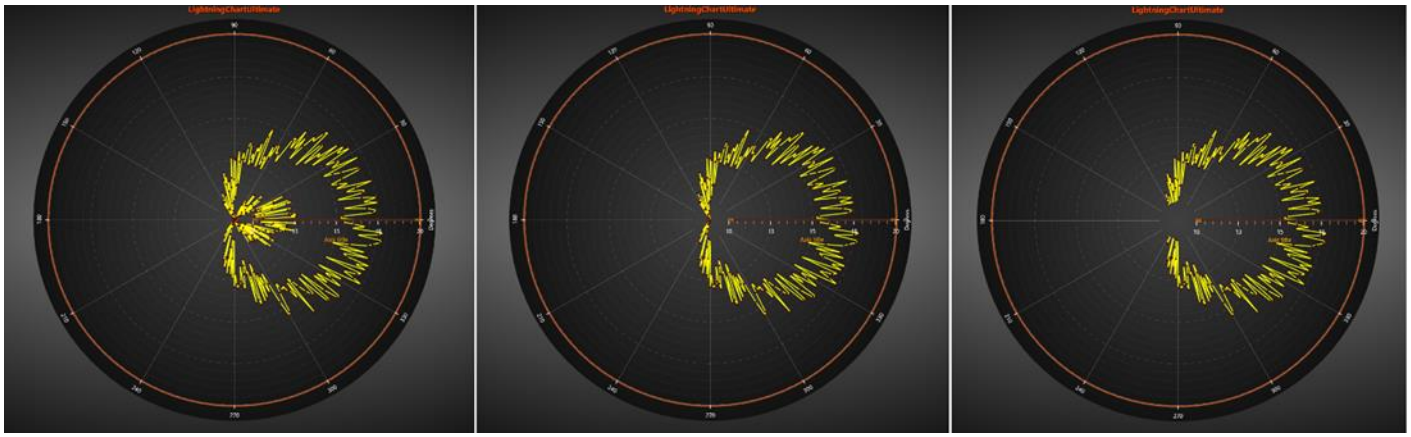


图 10-17. 左侧图 CenterClipping 设置为 None ,中间的图设置为 Center，右侧的图设置为 InnerCircle。 每一个图中 InnerCircleRadiusPercentage 设置为 10.

10. 14　自定义控件——半圆环图

演示示例：半圆环图

半圆环图，也称为半饼图，是一种自定义极坐标图，已预先配置为半圆环形状。 尽管常规极坐标图可用于创建类似的半圆环图，但使用此自定义控件可显着减少用户在实现这些图表时需要执行的步骤。

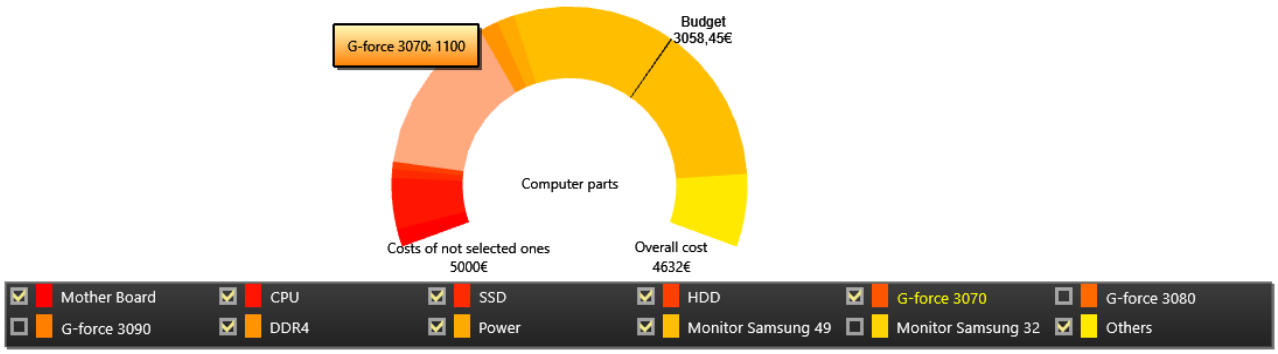


图 10-110-8. 半圆环图例

可以像 LightningChart 对象一样将半圆环图添加到应用程序中。 首先创建一个新的 Half Donut 对象，然后将其添加到容器元素中，例如 Grid。 在 BeginUpdate() 和 EndUpdate() 调用中配置图表以减少需要渲染的帧数。

```
HalfDonut donut = new HalfDonut();
(Content as Grid).Children.Add(donut);
donut.BeginUpdate();
// Configure half donut chart here
donut.EndUpdate();
```

10.14.1 添加数据

可以通过调用 AddSlice() 方法将数据添加到半圆环图。

```
// Adding a slice to a donut.
donut.AddSlice(100, "Electricity", Colors.Yellow);
```

Individual slices can removed via ***RemoveSlice()*** -method.

```
// Removing a slice at index 1.
donut.RemoveSlice(1);
```

Alternatively, use ***HideSlice()*** to hide and show the slices.

```
// Hiding the first slice.
donut.HideSlice(0, false);

// Showing the first slice.
donut.HideSlice(0, true);
```

10.14.2 配置半圆环图

半圆环图有几个配置选项。 可以完全修改文本、开始和结束角度以及颜色。 针和标记线也可以调整或隐藏。

```
// Modifying texts
donut.Title = "Chart title";

// Using custom texts
donut.EndAndStartPointText = HalfDonut.EndAndStartPointTexts.Custom;
donut.CustomLeftSideText = "New text string";
donut.CustomNeedleText = "Needle text";

// Disable needle
donut.ShowNeedle = true;
```

圆环图共有三种着色等级：HSV、HSVA 和 Slice。 使用 HSV 或 HSVA 着色时，调整各个通道，然后使用 ColorStep 修改切片之间的颜色差异。

```
// Setting colors.
donut.SelectedColorPalette = HalfDonut.ColorScale.HSVA;
donut.ColorStep = 5;
donut.Saturation = 0.9;
donut.StartingColorValue = 30;
donut.Value = 0.6;
donut.Alpha = 0.4;
```

当 ColorScale 设置为 Slice 时，应用通过 AddSlice() 添加切片时分配的颜色。

GetInternalChart() 方法可用于访问内部 LightningChart 组件。这适用与 当圆环图本身的属性不够全面时。

```
// Modifying the legend box.
donut.GetInternalChart().ViewPolar.LegendBox.Fill.Color = Colors.SkyBlue;
```

或者，使用各种 Get 方法，这些方法允许直接访问和修改内部 LightningChart 组件，例如注释。

```
// Changing the internal annotation object.
donut.GetLeftSideText().TextStyle.Color = Colors.LimeGreen;
```

10.14.3 HalfDonutControlPanel

HalfDonutControlPanel 是一个 UI 元素，旨在在应用程序运行时修改半圆环属性。 将控制面板添加到应用程序的工作方式类似于添加半圆环图。 创建一个新的 HalfDonutControlPanel 对象并将其添加到 UI 容器中。 之后，可以向控制面板添加一个或多个半圆环对象。 然后可以使用该面板来修改添加的圆环图。

```
// Creating a HalfDonutControlPanel
HalfDonutControlPanel cp = new HalfDonutControlPanel();

// Adding to a grid element
_controlGrid.Children.Add(cp);

// Add an existing half donut object.
cp.HalfDonuts.Add(donut);
```

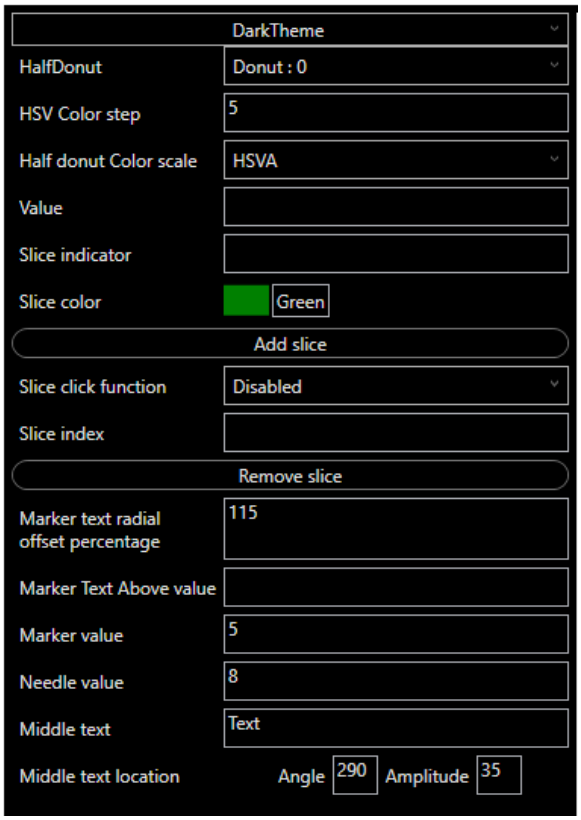


图 10-19. 选择了暗色主题的 *HalfDonutControlPanel*

11. ViewSmith 史密斯圆图视图

史密斯图通常用于阻抗测量和阻抗匹配的应用程序中。

史密斯图表用实值和虚值绘制数据（**R + jX**）.

Terms

Impedance = **Z** = **R** + j**X**

R = Resistance, Real part

X = Reactance, Imaginary part

X > 0: Capacitive
X < 0: Inductive

在圆形实重对数和虚重对数刻度上，数据位置是按角度绘制2D图决定的。

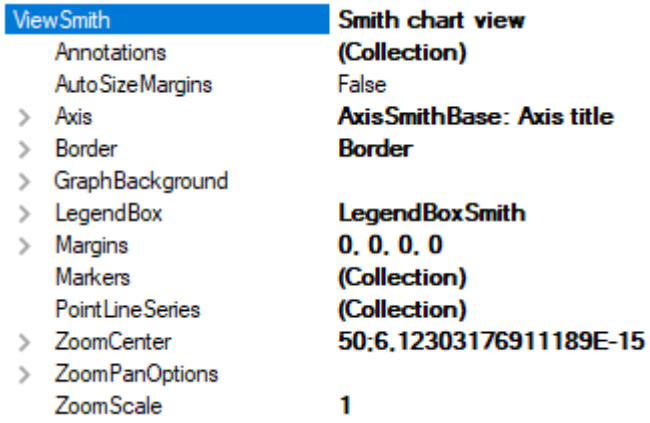


图 11-1. ViewSmith 属性树

11.1 Axis 轴

史密斯图表只有一个实轴，通过扩展属性树 ***Axis*** 可以对其进行配置，参阅第 10-2 章节。

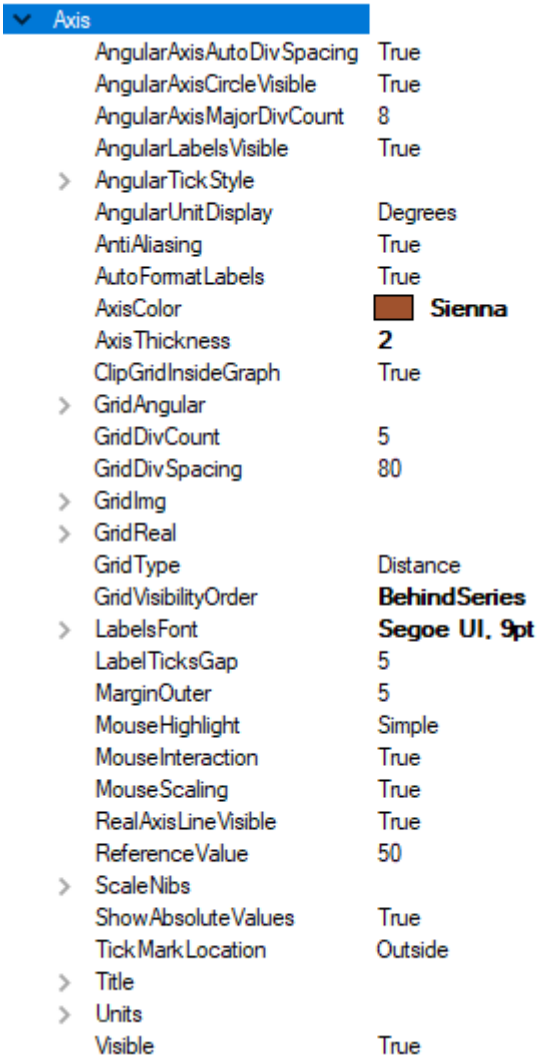


图 11-2. 史密斯图轴属性树

大多数属性与 ViewPolar 的轴以及 ViewXY 的轴相同，可以自定义并使图表更具美观。针对 ViewSmith 调整还有一些高级属性，例如 ***GridDivCount***、***GridImg*** 和 ***GridReal***、***RealAxisLineVisible***、***ShowAbsoluteValues***、***ClipGridInsideGraph***。

GridDivCount 定义了实轴上的圆形网格线数量以及虚标度上的对数网格线的数量。

GridImg 和 ***GridReal*** 属性负责在 ***Real*** 或 ***Imaginary*** 刻度上自定义网格线。此外，***Visible*** 属性可用来隐藏网格，因此用户可以隐藏其中的一个，然后继续用其他的来工作。

RealAxisLineVisible 属性可用以隐藏轴线。

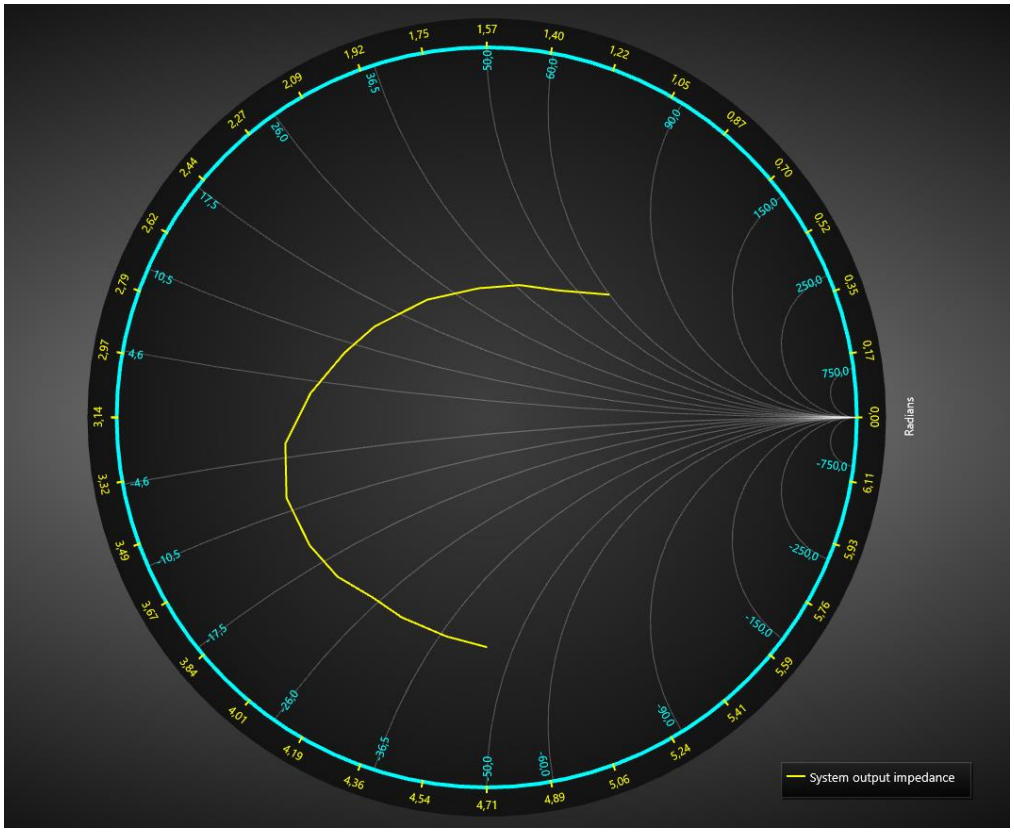


图 11-3. 隐藏实网格线，虚网格线可见。

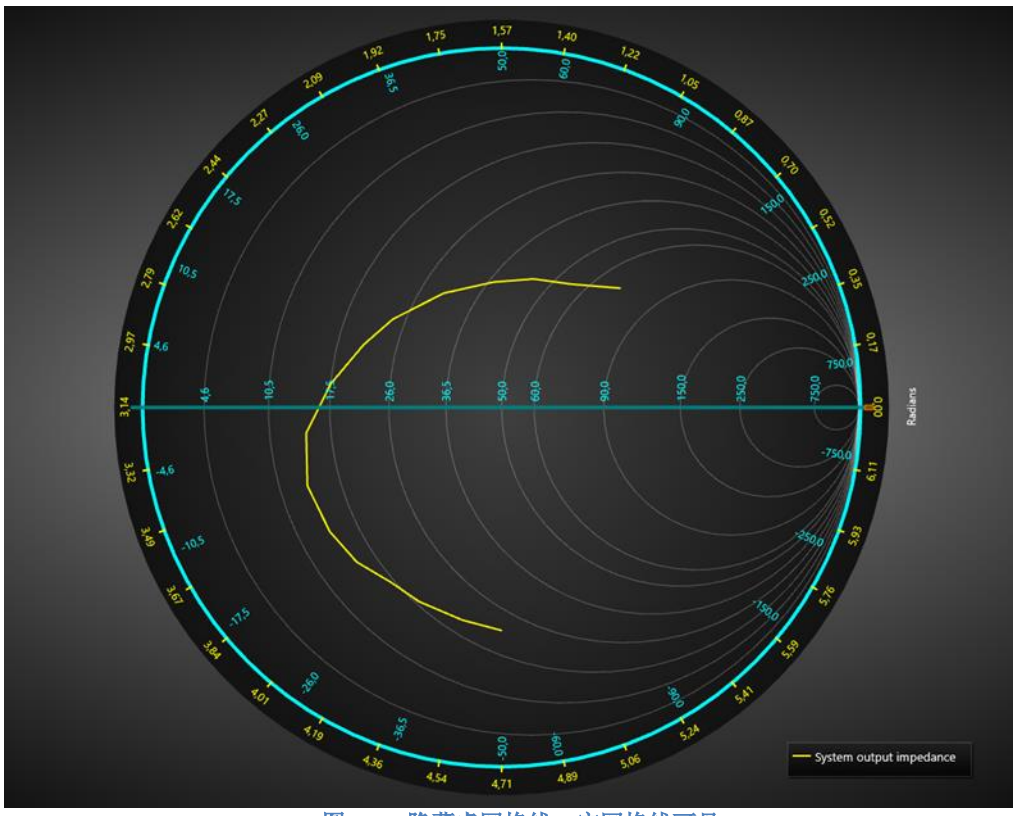


图 11-4. 隐藏虚网格线，实网格线可见。

ShowAbsoluteValues 属性定义了将在刻度上显示的值（绝对值或标准值）.

ClipGridInsideGraph 可以让网格线在图表圆之外可见。

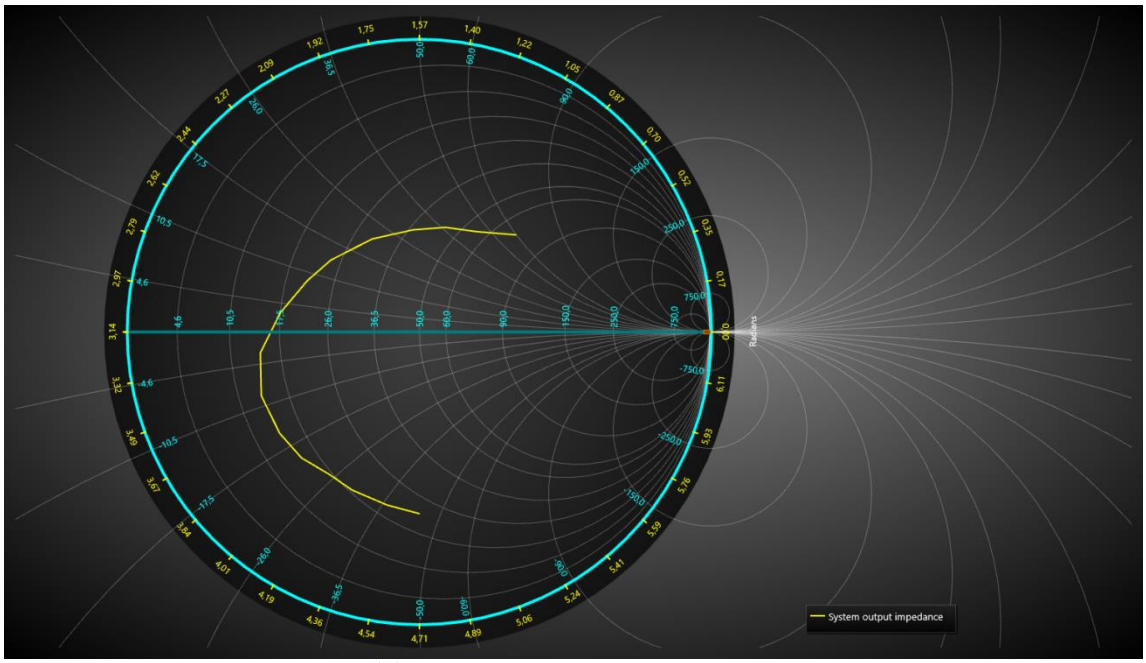


图 11-5. ClipGridInsideGraph = False.

完全自定义的史密斯圆图表如下所示。

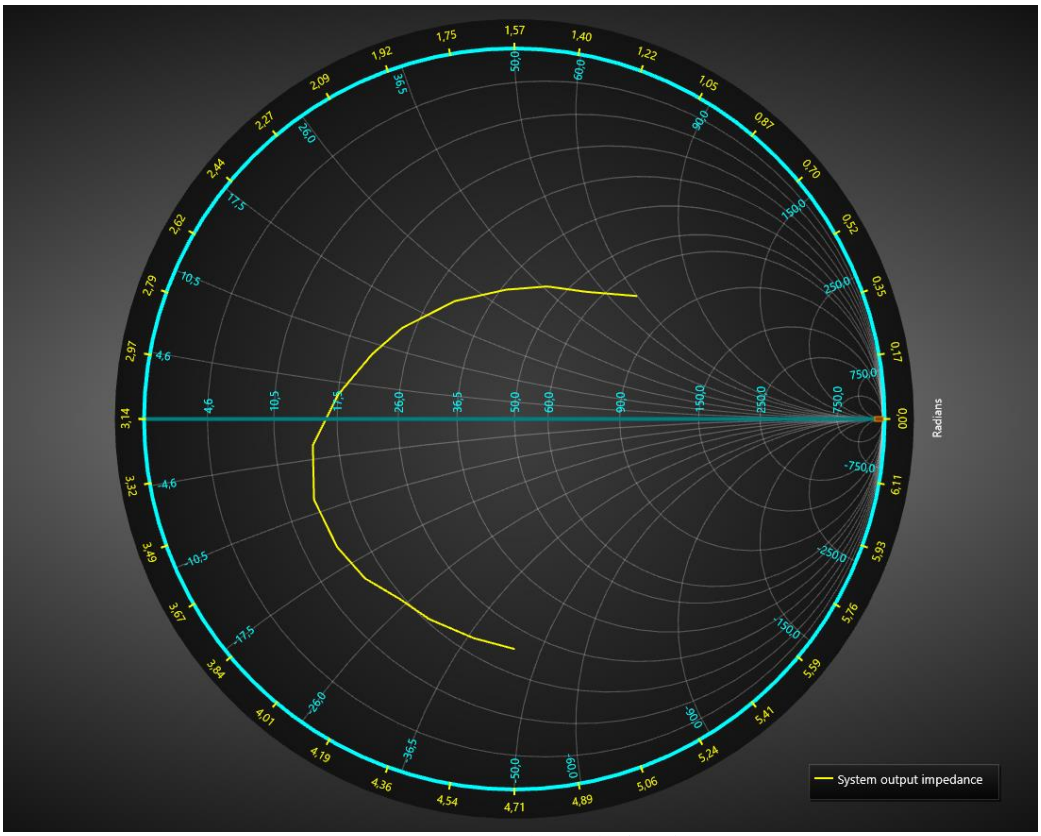


图 11-6. 自定义的史密斯圆图

11.2 图边距

当设置 ***AutoAdjustMargins*** 为 ***enabled*** 后，可以调整图表大小来腾出足够的空间放置所有的轴与图表标题。当设置其为 ***disabled*** 后，应用 ***ViewSmith.Margins*** 属性可以手动设置图边距。

在运行时，可以通过调用 ***ViewSmith.GetMarginsRect*** 方法以像素为单位检索图边距矩形，该方法既适用于自动图边距，也适用于手动图边距。当需要进行基于屏幕坐标的计算或对象布置时，这非常有用。

ViewSmith.MarginsChanged 事件可设置为在更改图边矩矩形（例如调整其大小）时触发。

视图的内容在图边距之外的部分会自动裁剪掉。除了图表标题、注释和图例框之外的其他所有内容都会被剪切掉，这是因为它们的位置是以屏幕坐标定义的，使得它们可以任意地放置在图边距上。 还可以绘制一个 1 像素宽的边框矩形 ***Border***,来显示图边距的位置。默认情况下，在 ViewSmith 视图中该边框不可见。此矩形的颜色可以通过 ***Border.Color*** 来变更。

11.3 图例框

图例框在 ViewSmith 中的工作方式与 在 ViewPolar （参阅第 **Error! Reference source not found.**章节）中完全一样。通过 ***ViewSmith.LegendBox*** 可以修改图例框属性。

11.4 PointLineSeries

演示示例： Line and data cursor

ViewSmith 的 ***PointLineSeries*** 和在 ViewPolar 视图中的一样，可用来绘制一条线、一组点或一个点线。在 ***LineStyle*** 和 ***PointStyle*** 属性中，有许多线与点的样式可供选用。

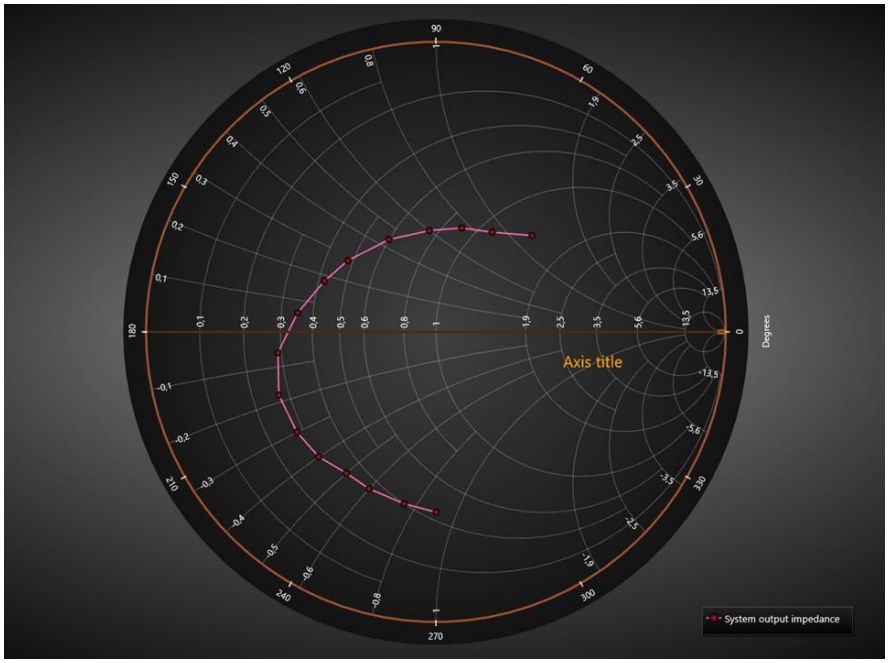


图 11-7. 史密斯图数据系列

11.5 设置数据

下面的代码，将向史密斯图表的集中添加一组数据点。

```
SmithSeriesPoint[] m_aPoints;
PointLineSeriesSmith Series = new PointLineSeriesSmith（m_chart.ViewSmith, axis）;
// 为系列创建数据
m_iCount = 5000;
m_aPoints = new SmithSeriesPoint[m_iCount];
for （int i = 0; i < m_iCount; i++）
{
          //从左到右的正弦
          m_aPoints[i].RealValue = i * （MaxReal / m_iCount）;
          m_aPoints[i].ImgValue = Math.Sin（0.01 * i）/Math.PI * MaxReal;
}
Series.Points = m_aPoints;
// 向图表添加系列
m_chart.ViewSmith.PointLineSeries.Add（Series）;
```

11.6 注释

Annotations 与 ViewXY 的 ***Annotations***（参阅第 6.26 章节）类似，但以史密斯图轴值（实与虚）定义的 ***Target*** 和 ***Location*** 除外。***Sizing*** 属性仅具有 ***Automatic*** 和 ***ScreenCoordinates*** 值。

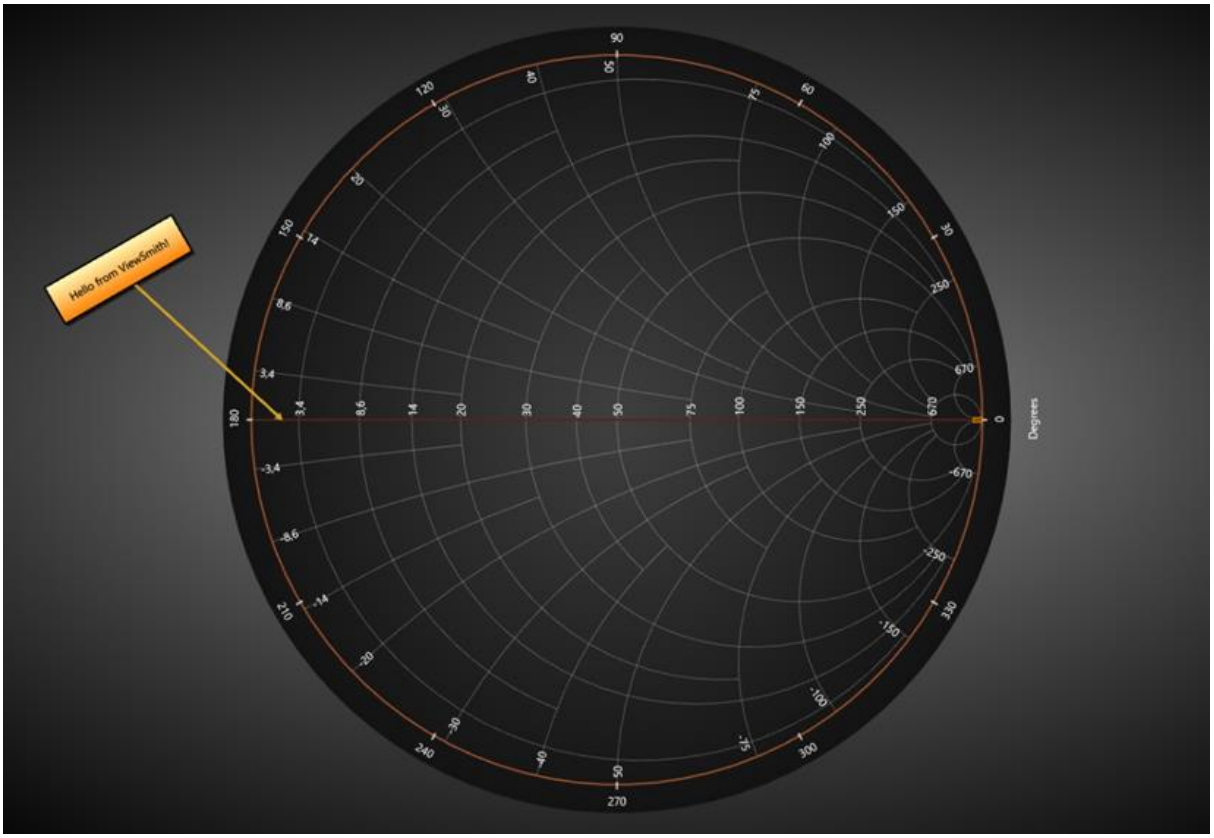


图 11-8.在 ViewSmith 中的一个注释示例

11.7 标记

演示示例： Line and data cursor

标记符可以用来标记在某特定位置的特殊数据。标记可以通过用鼠标拖动来移动。此属性与 ViewPolar 的标记（参阅第 **Error! Reference source not found.**章节）具有相同的定义 。

定义 ***ImgValue*** 和 ***RealValue*** 属性可以进行布置。编辑 ***Symbol*** 可设置首选外观，用 ***Label*** 属性可以定义文本内容。

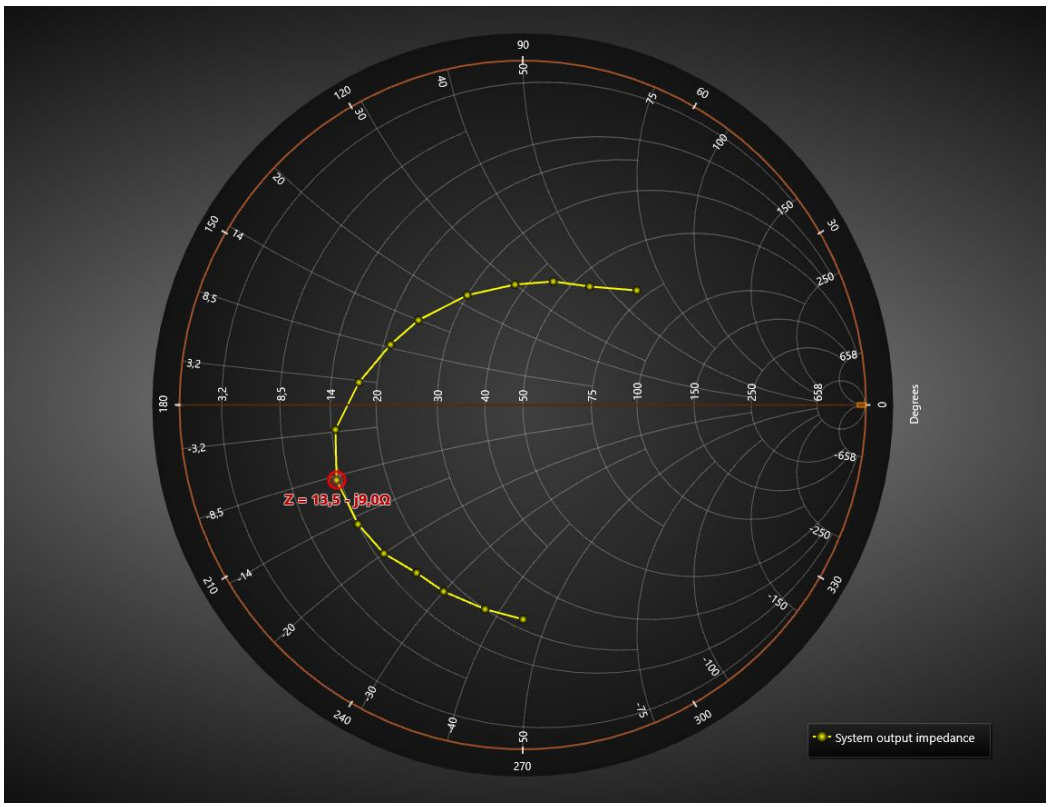


图 11-9.在史密斯圆视图中一个标记跟踪一个系列

11.8 数据游标

从版本 10.5 开始，ViewSmith 有一个内置的数据游标，它会自动跟踪最接近鼠标光标的系列值，并允许在结果表中显示它。 光标的工作方式与 ViewPolar 中的光标相同，除了显示实数值和虚数值而不是振幅和角度。 请参阅第 10.9 章了解如何配置数据游标设置。

数据游标跟踪 PointLineSeries 但不跟踪标记。

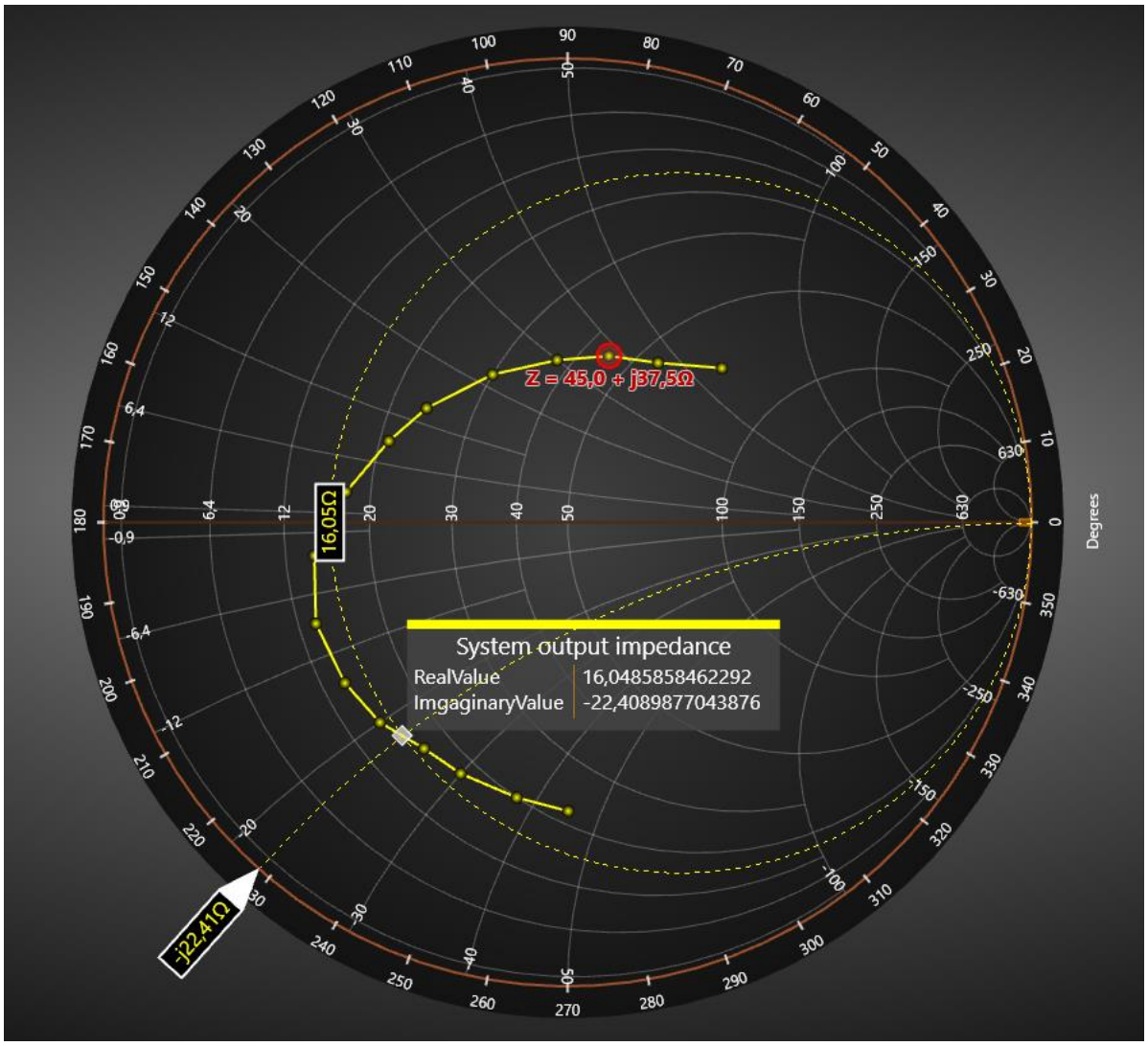


Figure 11-10. ViewSmith 中的数据游标

11.9 缩放与平移

在 ViewSmith 中缩放与平移选项以及方法的运行原理与在 ViewPolar （参阅第 **Error! Reference source not found.**章节）完全一样。

12. 设置颜色主题

用 ***ColorTheme*** 属性可以设置图表的整体颜色主题 。设置主题将覆盖创建的图表中的大多数对象颜色。 建议首先设置 ***ColorTheme***，然后修改对象颜色。

```
chart.ColorTheme = ColorTheme.SkyBlue; // Changing the color theme
```

注意! 通过设置颜色主题，每个手动分配的颜色都将在 Visual Studio 属性网格中丢失，且不会有预先警告。

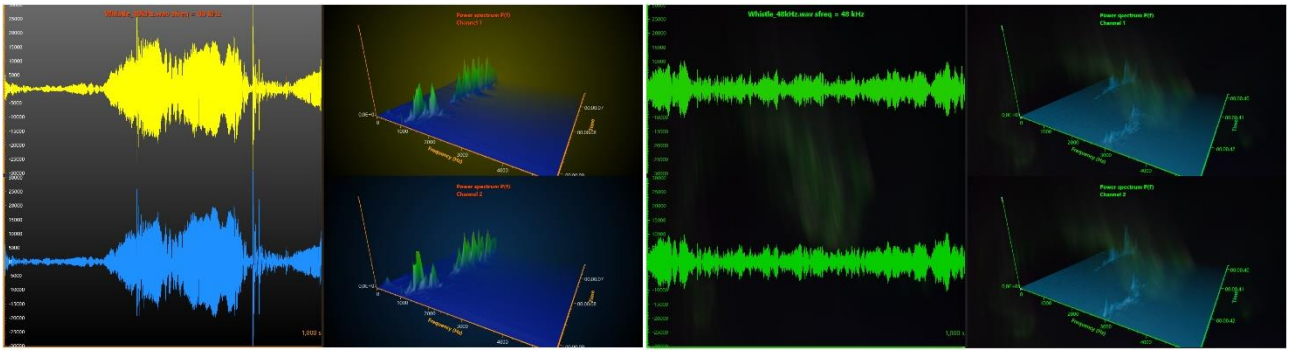


图 12-1.两个不同颜色主题。左侧是带有一些自定义颜色的默认暗色主题；右侧是光环主题设置。

12.1 自定义主题

LightningChart 允许通过更新 **CustomDynamicTheme -property** 创建自定义颜色主题。 有两种方法可以做到这一点。 将自定义主题设置为变量并更新，或者将 **ColorTheme** 设置为 **CustomDynamicTheme** 并更新其属性。

```
    // Method 1
    ThemeBasics theme = _chart.CustomDynamicTheme;
    theme.BackgroundColor = Colors.Black;
    theme.ChartTitleColor = Colors.Green;
    _chart.CustomDynamicTheme = theme;

    // Method 2
    _chart.ColorTheme = ColorTheme.CustomDynamicTheme;
    _chart.CustomDynamicTheme.ChartTitleColor = Colors.Green;
_chart.UpdateCustomTheme();
```

一些主题对轴和系列使用自动着色。 要手动设置轴或系列颜色，请分别禁用 **MultiColorAxis** 或 **MultiColorSeries**，然后设置颜色。

```
ThemeBasics theme = _chart.CustomDynamicTheme;
theme.MultiColorAxis = false;
theme.AxisColor = Colors.Blue;
_chart.CustomDynamicTheme = theme;
```

交互式示例演示应用程序具有 ThemeCreator，它允许研究颜色主题并创建新主题。

13. 滚动条

演示示例： Scroll bars; Historic data review; Scale breaks

通过 ***HorizontalScrollBars*** 或 ***VerticalScrollbars*** 集属性可以添加一个或多个滚动条。其外观可以完全自主定义，甚至可以定义椭圆形的按钮与滚动块。例如，可以将一个位图用做按钮图标。滚动条可以用于所有视图中，但最醒目的用途是在 ViewXY 中。

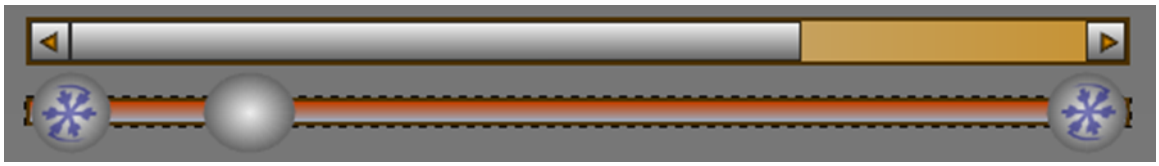


图 13-1. 两种不同外观的滚动条

13.1 滚动条属性

通过设置 ***Alignment*** 属性为 ***BelowGraph***、***AboveGraph*** 或 ***GraphCenter***，可以将 ***HorizontalScrollBar*** 对齐调整以适应图形的宽度。另外，通过设置 ***Alignment*** 属性为 ***LeftToGraph***、***GraphCenter*** 或 ***RightToGraph***，可以将 ***VerticalScrollBar*** 对齐调整以适应图形的高度。设置 ***Alignment*** 为 ***None***，用 ***Offset*** 属性可以任意放置滚动条。用 ***Size*** 属性则可以调整其尺寸大小。

```
// 设置一个滚动条的位置与大小。Offset是基于图表的左上角开始计算。
horizontalScrollBar.Alignment = HorizontalScrollBarAlignment.None;
horizontalScrollBar.Offset = new PointIntXY（100, 10）;
horizontalScrollBar.Size = new Size（500, 30）;
```

滚动条使用的是 64 位无符号整数值，而不是通常的 32 位有符号整数值。***Value*** 即为当前的位置， ***Minimum*** 是最小的值域值， ***Maximum*** 是最大的。这便可以以高采样频率直接支持长整形测量值。 例如，当在测量中使用 ***SampleDataSeries*** 时，将采样索引直接设置为滚动条值。***Minimum*** 值表示第一个采样索引， ***Maximum*** 表示最后的采样索引。

SmallChange 属性是单击滚动按钮时增加或缩减的量。如果 ***KeyControlEnabled*** 生效，并且滚动条具有焦点，则可以使用箭头键按 ***SmallChange*** 量更改 ***Value***。***LargeChange*** 表示在滚动块或滚动按钮外单击滚动条时发生的页面变更。用 ***PageUp*** 和 ***PageDown*** 键可分别改变 ***Value***。用 ***MouseWheelChange*** 可以设置当鼠标滚轮在滚动条上滚动时的变更值。

在代码中可以使用 ***Scroll*** 事件处理程序来响应滚动条值的变更，或者可以使用 ***ValueChanged*** 事件处理程序。不过，***Scroll*** 提供了有关如何完成滚动的更多信息。

13.2 带有小数或负值的滚动条

由于滚动条的设计目的在于使用无符号整数来支持长整形测量值，所以当轴值是小数或负值时不能直接使用它们。 在这些情况下，由于四舍五入错误，结果将导致崩溃或其他不良可用性。 但是，滚动条的 ***Minimum*** 和 ***Maximum*** 值不必绑定到轴范围。这样，即使轴值和（或）数据值是较小的小数或负数，也可以缩放滚动条。

例如，如果要显示的从 0.500 到 1.500 的值范围，则可以将滚动条的使用范围设置为 500 -> 1500。另外，使用-150 到 0 之间的数据范围，滚动条可以使用 0 到 150 之间的值。

下面的示例演示如何当 y 轴值是从负到正的小数时，对垂直滚动条进行修改。

```
double yMin = -0.178; // 某个任意轴值
double yMax = 1.253;
double upShift = 0.178; // 防止滚动条使用负值
double scaleFactor = 1000.0; // 缩放滚动条以使用整数值
_chart.ViewXY.YAxes[0].Minimum = yMin;
_chart.ViewXY.YAxes[0].Maximum = yMax;
_chart.VerticalScrollBars[0].Minimum = (ulong)((yMin + upShift) * scaleFactor
);
_chart.VerticalScrollBars[0].Maximum = (ulong)((yMax + upShift) * scaleFactor
);
_chart.VerticalScrollBars[0].LargeChange = _chart.VerticalScrollBars[0].Maximum
          - _chart.VerticalScrollBars[0].Minimum;
_chart.ViewXY.YAxes[0].SetRange(0.3, 0.6); // 只显示轴的一部分


// 滚动事件
private void VerticalScrollBar_Scroll(object sender, ScrollEventArgs e)
{
    double newMin = (yMax + upShift) * scaleFactor - (double)e.NewValue -
           currentRangeY + (yMin + upShift) * scaleFactor;
    if (newMin < (yMin + upShift) * scaleFactor)
    {
        // 滚动条不能在低于最小轴值时运行
        newMin = (yMin + upShift) * scaleFactor;
    }

    double newMax = newMin + currentRangeY;
    if (newMax > (yMax + upShift) * scaleFactor)
    {
        //滚动条不能在大于最大轴值时运行
        newMax = (yMax + upShift) * scaleFactor;
        newMin = newMax - currentRangeY;
    }

// 根据滚动条值调整轴范围，触发RangeChanged事件。将滚动条使用的值转换回未缩放的轴值
```

```
    _chart.ViewXY.YAxes[0].SetRange((newMin / scaleFactor - upShift),      (
newMax / scaleFactor - upShift));
}

// RangeChanged -事件，正确修改轴范围
private void AxisY_RangeChanged(object sender, RangeChangedEventArgs e)
{

    // 防止轴最小值小于之前设置的yMin
    if (e.NewMin < yMin)
    {
        double newRange = e.NewMax - e.NewMin;
        if (newRange <= yMax - yMin)
        {
            _chart.ViewXY.YAxes[0].SetRange(yMin, yMin + newRange);
        }
        else
        {
            _chart.ViewXY.YAxes[0].SetRange(yMin, yMax);
        }
    }
    // 防止轴最大值大于之前设置的yMax
    else if (e.NewMax > yMax)
    {
        double newRange = e.NewMax - e.NewMin;
        if (newRange <= yMax - yMin)
        {
            _chart.ViewXY.YAxes[0].SetRange(yMax - newRange, yMax);
        }
        else
        {
            _chart.ViewXY.YAxes[0].SetRange(yMin, yMax);
        }
    }
    // 根据新轴范围修改滚动条
    else
    {
        double newRange = (e.NewMax - e.NewMin) * scaleFactor;
        _chart.BeginUpdate();

        _chart.VerticalScrollBars[0].LargeChange = (ulong)(newRange + 0.1);
        _chart.VerticalScrollBars[0].Value = (ulong)((yMax - e.NewMax +
          yMin + upShift) * scaleFactor);

        _chart.EndUpdate();

        currentRangeY = newRange;
    }
}
```

14. 导出与打印

14.1.1 位图图像导出

图表用 ***SaveToFile*（）**方法可以导出为.PNG、 .BMP 和.JPG 格式文件。 采用 ***SaveToFile*（…）**方法可以利用分辨率递减和平滑/反锯齿选项来导出图像文件。使用方法可以导出一个流（stream）。

14.1.2 矢量图像导出

ViewXY、ViewPolar 和 ViewSmith 也能够以.WMF、.EMF 和.SVG 格式导出。View3D 和 ViewPie3D 现在并不支持。 使用 ***SaveToFile*** 或 ***SaveToStream*** 方法可用到所选择得矢量文件格式。

注意! 矢量导出简化了，而且所有的细节，比如复杂的点样式等，可以用普通又简单的颜色和形状表示。矢量导出也可能包含一些位图元素。

14.1.3 复制到剪贴板

通过调用 ***CopyToClipboard*（…）**可以将图表复制到剪切板。ViewXY、 ViewPolar 和 ViewSmith 能够以矢量格式用 ***CopyToClipboardAsEmf*（）**方法进行复制。

14.1.4 捕获至字节数组

图表的 ***CaptureToByteArray*** 方法，可获取作为快速原始图像的数据复制到外部组件或进一步处理的数据。

用途

```
int width;
int height;
PixelFormat format;
byte[] aData = _chart.CaptureToByteArray(out width, out height, out
format);

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);
```

14.1.5 设置连续帧写入的导出流

使用 ***chart.OutputStream*** 属性设置一个流，图表会将其渲染的帧写入其中。

此属性用于以最快的方式捕获图表中的连续帧，特别是在 ***Headless mode*** 模式下（请参阅第 25 章）。

该流是一个原始的字节流，每个像素用 4 个字节表示，每个信道一个字节。信道的顺序取决于渲染器及其设置。

使用 ***GetLastOutputStreamFormat*** 和 ***GetLastOutputStreamSize*** 方法可找出最后写出的图像的格式和导出尺寸。

生成的图像大小应该是图表的大小（以像素为单位）。

注意!与 LightningChart 的其他属性相反，所设置的流不会在图表的处理时进行处理。

注意!与其他属性相反，设置此属性不会引发渲染新帧。

14.1.6 打印

调用 ***PrintPreview（）*** 方法可以打开一个打印预览对话框，或用 ***Print（）*** 可以直接采用默认设置来打印。调用 ***Print（…）*** 可采用手动设置来打印。打印 ViewXY、 ViewPolar 和 ViewSmith 时，也支持矢量打印。***Print（…）*** 方法提供有 ***Raster*** 或 ***Vector*** 格式的打印参数。

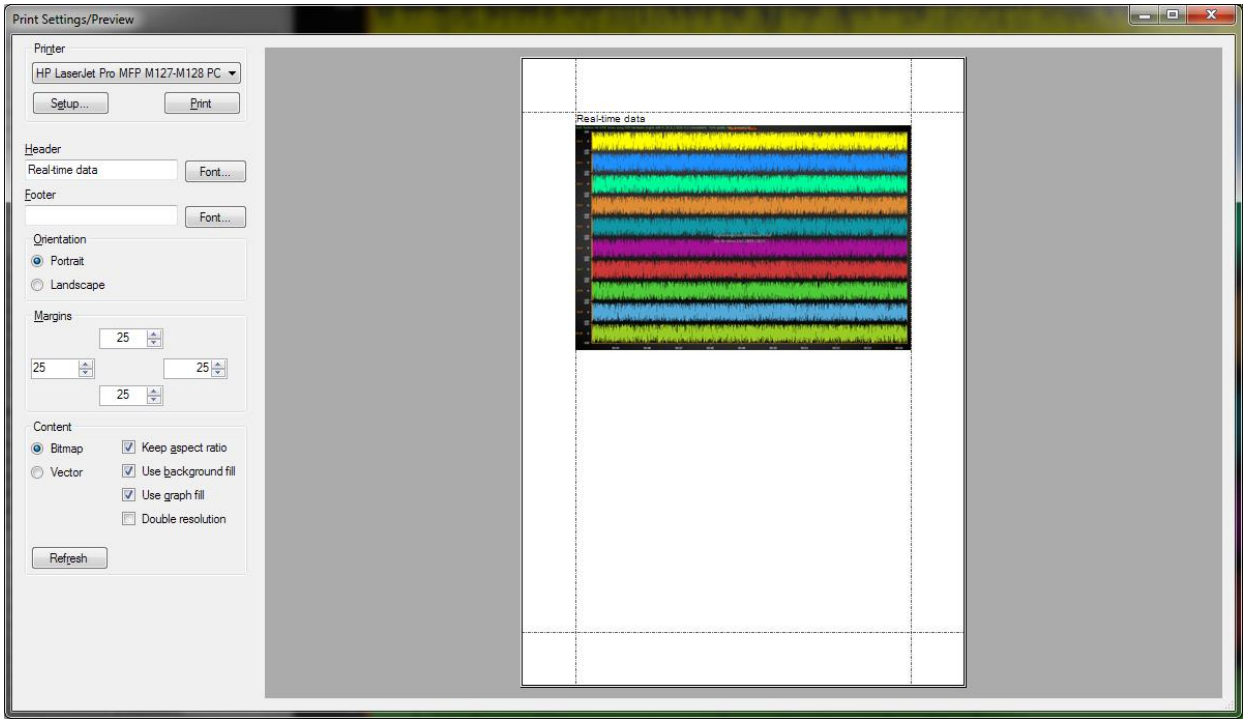


图 14-1. 打印预览对话框

15. LightningChart 性能

15.1 选择正确的 API 版本

根据第 1.1 章节的指示说明选择图表版本。除非必要，否则不要使用系列数据绑定特性。

15.2 正确设置渲染选项

在一些特定的应用程序中，LightningChart 的 DirectX9 渲染引擎可能会比 DirectX11 引擎稍微块些，但是一般来说，保留 DirectX11 作为首选的渲染程序是一个不错的选择。而且 DirectX11 还带来了更佳的外观。

字体特征的设置也很重要。

参阅第 5.121 章节。

15.3 更新图表数据或属性

每个属性或系列数据值发生变更，都将引发重新绘制 LightningChart 控件。每次重绘都会引发 CPU 和显示适配器的额外负担。如果以编程方式同时更改了多个属性，则应在 ***BeginUpdate*** （）和 ***EndUpdate*** （）方法调用之间批量更改属性。***BeginUpdate*** （）将在调用了 ***EndUpdate*** （）之后才开始绘制控件。 在挂起 ***BeginUpdate*** （）调用时还有一个内部计数器可用，当达到相等数量的

EndUpdate（）调用时，***EndUpdate（）***就重新绘制控件。下面的示例演示如何以最小的计算机负载来更新图表。

```
chart.BeginUpdate () ; //禁用重绘（redraws）

//添加数据至系列中
chart.ViewXY.SampleDataSeries[0].AddSamples（multiChannelSampleStream[0],
    false）;
chart.ViewXY.SampleDataSeries[1].AddSamples（multiChannelSampleStream[1],
    false）;
chart.ViewXY.SampleDataSeries[2].AddSamples（multiChannelSampleStream[2],
    false）;

//更新点计数器条
chart.ViewXY.BarSeries[0].SetValue（0,1,（double）totalPointsCollected,””,
    false）;

// 点计数器标签
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 () ; // 开启 redraws 并重绘
```

内部计数器可以使用嵌套更新如下:

```
void MainMethod（）
{
    chart.BeginUpdate () ;
    chart.Title.Text = “My title”;
    chart.ViewXY.XAxes[0].AxisColor = Colors.Red;

    UpdateSeriesColors（）;

    chart.EndUpdate () ;
    // 只重绘一次。
}

private void buttonCreate_Click（object sender, EventArgs e）
{
    UpdateSeriesColors（）; // 只重绘一次。
}

void UpdateSeriesColors（）
{
   chart.BeginUpdate () ;

   foreach（PointLineSeries series in chart.ViewXY.PointLineSeries）
   {
            series.LineStyle.Color =  Color.Yellow;
   }
```

```
            chart.EndUpdate();
        }
```

更新系列数据取决于数据的存储方式。对于数组系列，***InvalidateData（）***方法必须在更新数组内容后调用，否则 UI 不会得到有关更改的通知。

ObservableCollection, 对于在 WPF 中全绑定模式的数据绑定非常有用，由于自动通知，它将在每个点或点的字段中进行自我更新。因此并不需要 ***InvalidateData（）***。但是，与数组或列表相比，ObservableCollections 会引发性能稍微降低，这在有大量数据点的情况下尤其明显。

15.4 线系列建议

- 使用线路系列时，如果适合应用程序，请使用 SampleDataSeries。 它是绘制速度最快的一种，并且不需要像其他线系列类型那样多的内存。 如果这不是一个选项，请优先选择 PointLineSeries 而不是 FreeformPointLineSeries 。 如果需要最少的视觉定制，请使用这些系列的“Lite”版本。 参见第 6.13 章。如果数据点不需要是可见的，则设置 ***PointsVisible*** 属性为 false。
- 用 ***LineStyle.Width*** 属性设置线宽度为 1 。
- 通过设置 ***LineStyle.Pattern*** 为 ***Solid*** 来采用实线样式
- 设置线系列的 ***LineStyle.AntiAliasing*** 为 ***None*** 来禁用反锯齿，并设置图表的 ***AntiAliasLevel*** 为 0。
- 通过设置系列的 ***MouseInteraction*** 为 false，来禁用所有鼠标交互。另外，通过设置图表的 ***chart.Options.MouseInteraction*** 为 false，来禁用掉全部图表的鼠标交互。
- 如果不需要图例框，则将其隐藏掉。所有系列的 ***ShowInLegendBox*** -属性也可以禁用掉。这些方法特别适用于图表中有很多系列的情况。

```
chart.ViewXY.LegendBoxes[0].Visible = false;
chart.ViewXY.PointLineSeries[0].ShowInLegendBox = false;
```

15.5 强度系列建议

适用于: ***IntensityGridSeries***、***IntensityMeshSeries***

- 如果系列数据无法持续更新，更改系列的 ***Optimization*** 属性为 ***StaticData***。如果数据每秒变更多次，那么 ***DynamicData*** 是更佳的选择。
- 用 ***Optimization***: ***DynamicValuesData*** 仅跟新 ***Data*** 数组的 ***IntensityPoint*** 结构的 ***Value*** 字段，并调用 ***InvalidateValuesDataOnly*** 方法来更新图表。采用这种方式，更新要快得多，因为不必重新计算该系列的几何结构。这仅适用于节点的数据 X 和 Y 值保持不变的应用程序，例如在热成像解决方案中。

适用于: ***IntensityGridSeries***

- 对于高分辨率热成像应用程序，为 ***IntensityGridSeries*** 开启 ***PixelRendering***。

- -对于快速更新的数据集，使用 ***SetValuesData*** 和 ***SetColorsData*** 方法代替 ***Data*** 属性来节省内存，可提高性能。

15.6 3D 正交视图建议

采用 ***View3D.Camera.Projection = Orthographic*** 而不是 ***OrthographicLegacy*** 可获得最大性能。特别是在缩放具有许多系列和数据的视图时，可以发现有所不同（参阅第 **Error! Reference source not found.**章节）。

15.7 3D 曲面系列建议

适用于: ***SurfaceGridSeries3D***, ***SurfaceMeshSeries3D***, ***WaterfallSeries3D***

- 如果不需要光反射和阴影，通过设置 ***SuppressLighting*** 为 false 来禁用照明。
- 如果用到轮廓线，使用 ***FastColorZones*** 或 ***FastPalettedZones***，而不是 ***ColorLines*** 或 ***PalettedColorLines***.

适用于: ***SurfaceGridSeries3D***

- 若有滚动数据（例如 3D-频谱或光谱图），采用 ***InsertRowBackAndScroll*** 和 ***InsertColumnBackAndScroll*** 方法来更新数据以及轴范围。

15.8 地图建议

适用于: ***ViewXY.Maps***

- 在 X 轴和 Y 轴范围保持不变，其他数据显示在地图上的典型情况下，将 ***ViewXY.Maps.Optimization*** 设置为 ***CombinedLayers***。这可以将地图图层渲染到相同的缓冲区图像中，从而实现更高效的渲染。
- 如果地图标题应该显示在 ***IntensityGrid*** 或 ***IntensityMesh*** 系列上，则设置 ***ViewXY.Maps.Optimization*** 为 ***None***。

15.9 硬件

为了完全获得 LightningChart 应用程序的最大性能，计算机硬件必须很强大。在许多应用程序中，显示适配器电源比 CPU 电源更重要。所以尽可能使用最新的显示适配器。DirectX 9.0c 级显示适配器也可以。“c”源自于 DirectX Shader Model 3，会被一些特效需要到。

GetRenderDeviceInfo（）方法用来确定所使用的显示适配器是否支持某些特性。特别是，如果返回的信息声明不支持 ***FastVertexFormat***，这对性能表现来说不是件好事。

注意! LightningChart 是一种 GPU 硬件加速图表。如果没有好的 GPU，运行性能可能会比在最佳的情况下差很多。**PassMark’s Video Card Benchmarks** 是比较不同 GPU 性能的好资源。**Benchmarks**

（http://www.videocardbenchmark.net/gpu_list.php）. 这种显卡比其他显卡的得分高 10 倍，在 LightningChart 的使用中也快 10 倍，但是刷新率的总体差异并没有那么大，因为不同的计算机硬件可能成为瓶颈。

16. LightningChart 通知、错误和异常处理

从第 8.4 版开始，LightningChart 将通过 ***ChartMessage*** 事件将图表中的消息发送给用户。这些消息会包含关于图表性能、使用不当、警告或错误的通知。 定义 ***chart.ChartMessage*** 事件的处理程序来监听消息。该事件包含一个 ***ChartMessageInfo*** 结构，可以保存消息中的信息。

在第 8.4 版之前，图表通过 ***ChartError*** 事件发送消息（现标记为 obsolete），它所包含的信息要比 ***ChartMessage*** 少。用户可以侦听 ***ChartError*** 事件而不是 ***ChartMessages***，并获得相同的基本信息，但是建议使用 ***ChartMessage***。

通过 ***ChartMessageInfo*** 的 ***MessageSeverity*** 属性可以得知消息的重要程度。根据消息的严重程度可以对其进行筛选。信息的几个重要程度为：

- ***Debug*** - 用户通常不感兴趣且不需要操作的调试信息。
- ***Information*** – 图表的不当使用，例如，使用无效的属性设置时，不应影响图表性能。通常不需要用户操作。
- ***Warning*** – 一些不当的图表使用。发生了一些图表的错误用法，例如使用了一个已处理的对象，这可能会导致图表出现一些小问题，比如性能降低。可能需要用户操作。
- ***RecoverableError*** – 发生了一个错误，图表本应该恢复。用户必须监听 ***ChartMessage*** 事件，否则具有此严重级别的消息将作为异常情况抛出。
- ***UnrecoverableError*** – 发生了一个错误，图表无法恢复。可能指示引入异常。用户必须监听 ***ChartMessage*** 事件，否则具有此严重级别的消息将作为异常情况抛出。
- ***Critical*** – 图表中出现了一个严重错误，往往会作为异常情况抛出。

MessageType 属性可说明消息的基本类型，而 ***Details*** 属性具有与之相关的更具体信息。所有可能的消息类型都可以在位于 LightningChart 名称空间中的 ***MessageType*** 枚举名称中找到。

更改 ***chart.Options.ChartMessageMinimumLevel*** 属性值可以将不想要的信息过滤掉。该属性只允许通过事件系统发送设置的最低级别和更高级别的消息。默认情况下设置为 ***MessageSeverity.Warning***。

异常情况将作为 ***ChartException*** 对象抛出，该对象包含 ***ExceptionInfo*** 结构，其中包含与 ***ChartMessage*** 事件类似的关于异常的 s 详细信息。 在一些情况中，图表可能抛出其他类型的异常，例如渲染引擎异常。如果用户希望图表针对所有 ***MessageSeverity.Warning*** 严重级别或更高的消息抛出异常， ***chart.Options.ThrowChartExceptions*** 属性需要设置为 ***true*** （此默认为 ***false***）。

建议始终订阅 ***ChartMessage*** 事件，以获得关于图表中的错误和未侦听消息的可能异常情况通知。 如果因图表出现任何问题需要支持，请确保在应用程序中有一个操作消息/异常处理程序，记录 ***ChartMessages*** 并将它们纳入支持请求。

17. ChartManager 组件

ChartManager 控制可以用来协调几个 LightningChart 控件的相互操作。向表单中添加 ***ChartManager*** 控件。接下来，将管理器控件分配给所有 LightningChart 控件的 ***ChartManager*** 属性。

17.1 图表互操作，拖放

ChartManager 支持在 WinForms 中，系列可以在图表间拖放。因技术原因，在 WPF 中不可用。

系列的 ***DisableDragToAnotherAxis*** 属性必须设置为 ***False*** 以开启拖动。默认设置为 ***True***。

轴的 ***AllowSeriesDragDrop*** 属性必须设置为 ***False***， 以防止拖动特定的轴。默认设置为 ***True***。

将鼠标移到要拖动的系列上，按下鼠标左键开始拖动。

拖动 Y 轴: 将该系列拖动到另一个图表的 Y 轴上并释放按钮。另一个图表获取该系列的所有权，并将该系列分配给目标 Y 轴。这也为这个系列分配了**第一个 X 轴。**

拖动 X 轴: 将该系列拖动到另一个图表的 X 轴上并释放按钮。另一个图表获取该系列的所有权，并将该系列分配给目标 X 轴。这也为这个系列分配了**第一个 Y 轴。**

17.2 内存管理改进

在某些非常实时的监控应用程序中，如果应用程序运行时 CPU 负载很高，.net 垃圾收集器（无用单元收集程序）就不能很好地释放未使用的内存。 垃圾收集器一次释放所有内存，在更新图表时造成明显的“停止”或“暂停”。要使图表更新更顺畅，可以启用 ChartManager 的 ***MemoryGarbageCollecting*** 属性。 通过这种方法可以使用独立的线程来更频繁地释放内存，而不用考虑 CPU 负载。 使用 ***MemoryGarbageCollecting*** 时建议与多核处理器一起使用，因为运行的线程将使 CPU 稍微负载。

18. LightningChart® Trader

Trader 库（LightningChart.Wpf.TradingCharts.dll / Arction.WinForms.TradingCharts.dll）包括多种控件、工具和方法，可以轻松创建交易和金融应用程序。The Trader 库是在稳健又快速的 LightningChart API 基础上建立的。

TradingChart 是目前一个主要的控件。TradingChart 提供了一个用于构建交易应用程序的属性和方法的简洁界面，消除了复杂工程应用程序产生的 API 开销。当前版本包括 WPF 和 WinForms 表单图表，UWP 交易图表将随后可用。

本文档演示了 LightningChart® TradingChart 的基本用法，及其大部分可用属性和设置。在 ***Interactive Examples*** 演示程序中也有一些基于 TradingChart 的实例。建议都浏览一下，以便对用 TradingChart 建立交易应用程序有更好的理解。

请注意，LightningChart® TradingChart 需要特殊许可证才能使用。 联系我们获取更多信息。

18.1 基本用法

18.1.1 创建 TradingChart

要使用 TradingChart，则要向项目添加相对应的程序集。首相，向项目的 References 中添加 Arction.Wpf.TradingCharts.dll 。

```
using LightningChartLib.WPF.Trader;
```

然后就可以创建 TradingChart 对象了。

```
// 创建 TradingChart 组件 component
TradingChart _chart = new TradingChart();


//将图表添加到父容器中，在本例中为网格。
(Content as Grid).Children.Add(_chart);
```

从 Visual Studio 的工具箱中拖拽，也可以创建 TradingChart 组件。如果电脑已经安装了 **LightningChart .NET SDK v.9** 或以上版本，则可以使用此方法。

以上两种方法的结果如下所示。

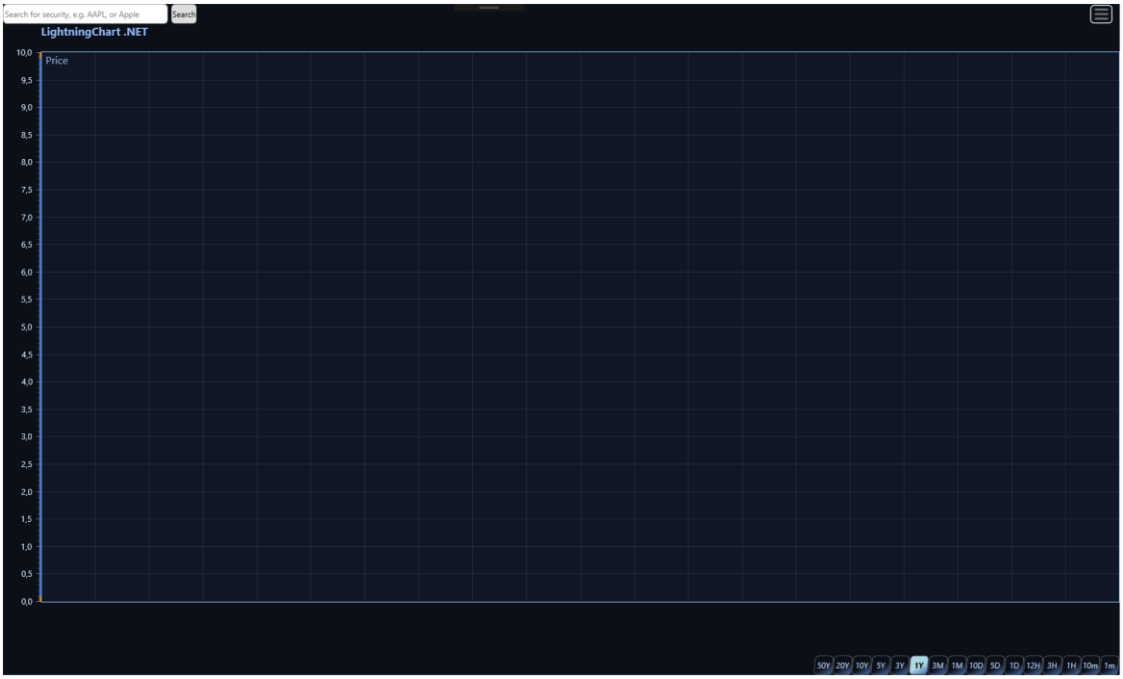


图 18-1. 已创建一个 TradingChart，并添加到容器中。

18.1.2 在 WinForms 应用中使用 TradingChart

自 LightningChart 10.0 版本开始，可以在 WinForms 应用中使用 TradingChart。通常来说，WinForms Trader 与 WPF Trader 的使用方法类似。这两个版本中所有的特性和属性都可以用。

用代码创建一个 WinForms TradingChart：

```
using Arction.CustomControls.Trader.WinForms;


// 创建 TradingChart 组件
TradingChart _chart = new TradingChart();

//将图表添加到父容器中，在本例中为网格。
_chart.Parent = this;
_chart.Dock = DockStyle.Fill;
```

18.1.3 部署 TradingChart

要在部署了该软件的计算机中运行 TradingChart 应用程序，必须通过代码应用部署密钥。这一步与普通 LightningChart（参阅第 4.4 章节）的方法类似，必须将对内部图表组件的引用添加到项目中 (*Arction.Wpf.Chart.LightningChart* 或 *Arction.WinForms.Chart.LightningChart*).

18.2 配置用户界面

TradingChart 有几个属性可用以控制用户界面的外观。

18.2.1 设置颜色主题

TradingChart 有几个预定义的颜色主题可供选择。通过 ***SetAppearance()*** 方法也可以进行修改。

```
// 设置主题色
_chart.SetAppearance(Appearance.Dark);
```

通过工具菜单可以更改颜色主题，默认情况下工具菜单在图表的右上角可以找到。

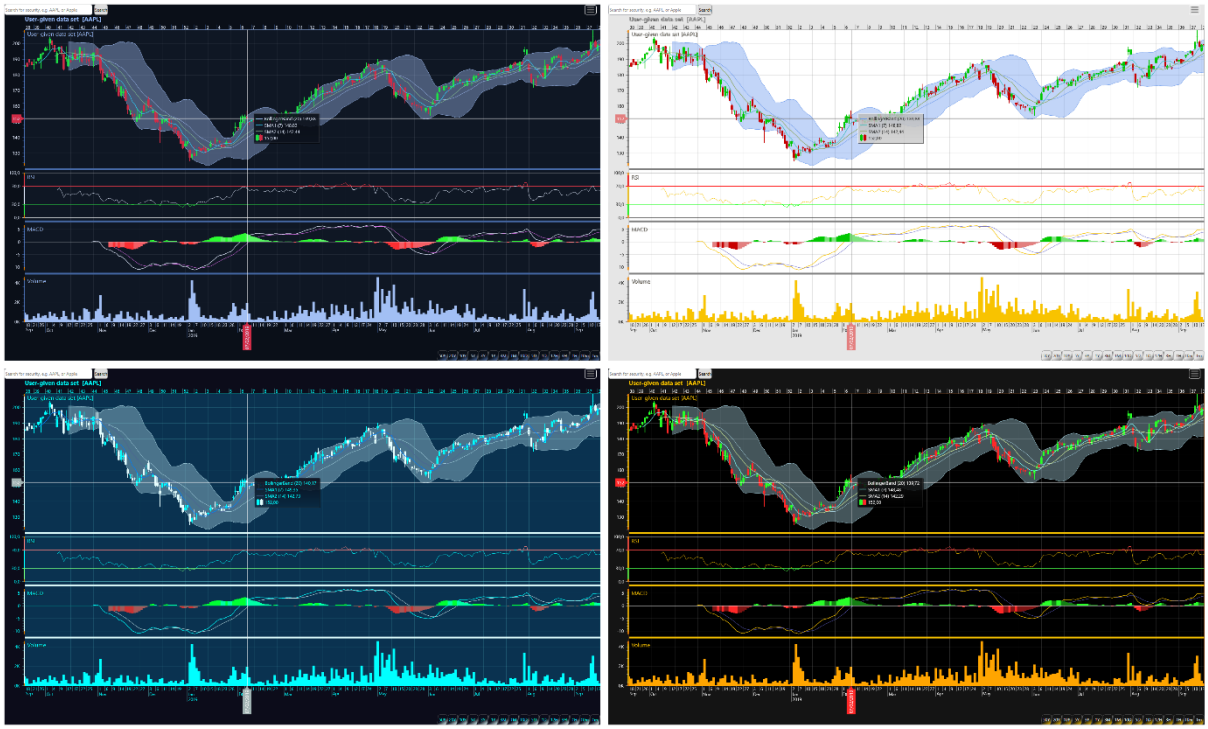


图 18-2. 部分预定义颜色主题

注意，更改颜色主题将覆盖掉为各指示和绘图工具手动设置的颜色。所以，只有在设置了颜色主题之后，才能对颜色进行单独修改。

18.2.2 设置价格图表样式

TradingChart 可以设置以何种样式显示交易数据。可选的样式 有：CandleSticks、Bars、Line 和 Mountain. 用 ***PriceChartType*** 属性可以进行样式变更。

```
// 交易数据显示为山型样式.
_chart.PriceChartType = PriceChartType.Mountain;
```

图 18-3. 可选用的价格图表样式

18.2.3 UI 组件

TradingChart 的用户界面有几个内置的组件，不需要时可以隐藏掉。默认情况下，所有组件都是可见的。

图 18-4. TradingChart 的主用户界面组件，都可以通过禁用相应的属性进行隐藏。

通过搜索栏可以根据代号或公司名称从供应商中（AlphaVantage.co）搜索交易数据。通过 ***ShowSearchBar*** 属性可以调控搜索栏的可见性。

```
// 隐藏搜索栏
_chart.ShowSearchBar = false;
```

工具菜单位于图表右上角。菜单中包括所有可选用的绘图工具（Drawing），还可以更改图表的颜色主题。该组件可以通过禁用 ***ShowToolMenu*** 属性来隐藏。

```
// 隐藏绘图工具以及颜色主题选项菜单
_chart.ShowToolMenu = false;
```

通过在图表右下角的按键可以对图表的时间范围（Time range）进行修改。用 ***ShowTimeRangeSelection*** 属性可以调节其可见性。注意，隐藏时间范围按键将自动调整底部边距，以删除图表下方不需要的空出区域。

```
// 隐藏时间范围按键.
_chart.ShowTimeRangeSelection = false;
```

像 ***Volume*** 和 ***RelativeStrengthIndex*** 等的一些技术指标绘制在图表下面的一个单独的图段中。这时，在各图段之间会自动绘制一条叫做图段切分线（Segment splitter）的垂直线。用鼠标拖动该图段切分线可以改变图段的高度。通过禁用 ***ShowSegmentSplitters*** 属性可以隐藏图段切分线。这也可以防止通过鼠标拖动修改图段的高度。

```
// 隐藏图段间绘制的垂线.
_chart.ShowSegmentSplitters = false;
```

18.3 使用内部 LightningChart 控件

TradingChart 是建立在标准的 LightningChart 控件之上。通过 ***GetInternalChart()*** 方法可以直接访问 TradingChartd 的内部 LightningChart 控件及其全部属性。因此，可能两个图表的特征兼而有之。若要访问内部图表，必须在项目中有含对各自程序集的引用（例如 LightningChart.Wpf.Charting.LightningChart）。

```
// 改变内部图表属性
LightningChart chart = _tradingChart.GetInternalChart();
chart.ViewXY.GraphBackground.Color = Colors.Black;

// 交替
_tradingChart.GetInternalChart().ViewXY.GraphBackground.Color = Colors.Black;


//向TradingChart添加标准的Annotation
LightningChart chart = _tradingChart.GetInternalChart();
AnnotationXY anno = new AnnotationXY(chart.ViewXY, chart.ViewXY.XAxes[0],
chart.ViewXY.YAxes[0]);
chart.ViewXY.Annotations.Add(anno);
```

要注意，改变 TradingChart 的属性可能会直接覆盖对内部图表控件所做的设置。例如，改变 TradingChart 的外观，将覆盖上面对 GraphBackground 的颜色设置。另外，例如在各自的集合中添加不同的系列这样的操作应该注意一下，因为 TradingChart 系列也使用相同的集合。

```
// 冲突情况的例子
LightningChart chart = _tradingChart.GetInternalChart();
StockSeries stockSeries = new StockSeries(chart.ViewXY, chart.ViewXY.XAxes[0],
chart.ViewXY.YAxes[0]);
chart.ViewXY.StockSeries.Add(stockSeries);


chart.ViewXY.StockSeries[0].Visible = false;
```

上面的例子中并没有更改新添加的 StockSeries 的可见性，而是将隐藏加载到 TradingChart 的 OHLC 数据，因为 TradingChart 及其内部图表使用相同的 StockSeries 集合。加载的 OHLC 数据使用 StockSeries 系列渲染，该系列保留了该集合的第一个索引。

18.4 添加交易数据

交易可以通过读取文件，从互联网数据提供商获取数据或通过 SetData（）方法在代码中设置来将交易数据添加到 TradingChart。 在后一种情况下，交易数据可以来自任何来源，只要它可以以 OHLC 格式呈现。

18.4.1 数据供应商

TradingChart 有一个内置的数据提供者，可以根据代号或证券名搜索相关证券。搜索栏通常在图表的左上角可见，禁用 ***ShowSearchBar*** 属性可以隐藏搜索栏。 要搜索证券，可以在文本框中输入搜索字符串，然后点击“Search”搜索按键或 Enter 键进行搜索。随后，搜索结果会在搜索栏下方列出。点击某一列结果，会从供应商加载相应的交易数据集，并添加到图表。获取数据时，会显示“Loading data set…”。加载过程所需时间根据供应商以及数据集的大小各有不同。

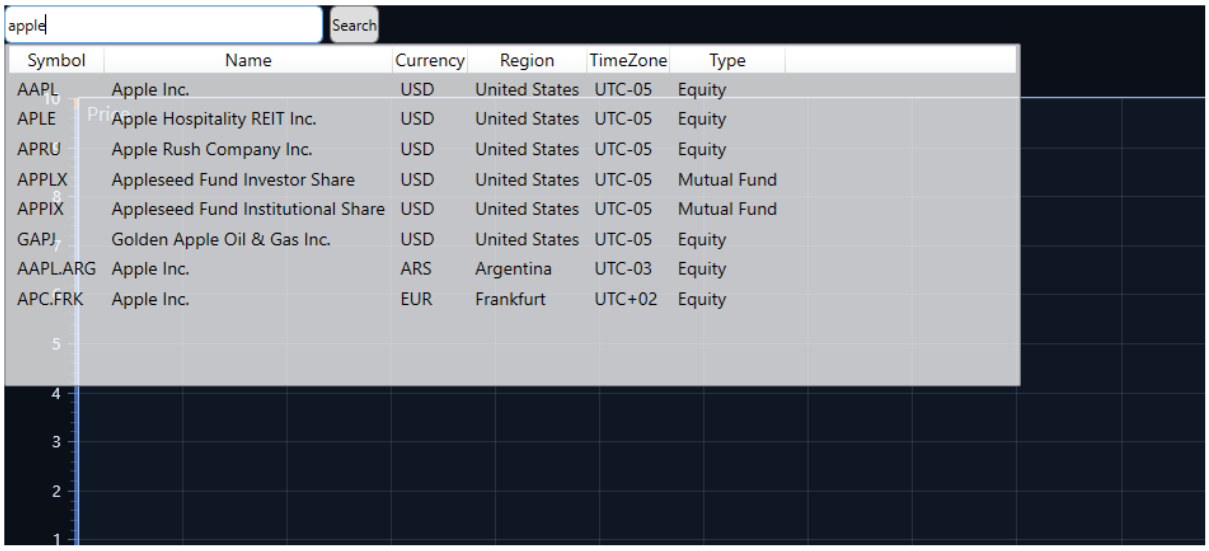


图 18-5. 基于搜索词“apple”的证券搜索。搜索结果分列在搜索栏的下方。

不使用搜索栏也可以获取数据。通过后台代码用 ***OpenSymbol()*** 方法，根据图表的 ***Symbol*** 属性获取数据。

```
//根据代号“GOOG”从互联网获取数据。

_chart.Symbol = "GOOG";
_chart.OpenSymbol();
```

通过 ***OpenSymbol()*** 可自动向图表添加已获得的数据。另外，可根据 ***Symbol*** 获取例如货币等数据，并在图表的标题中显示这些信息。

当向供应商请求数据时，通过 ***DataRequestType*** 属性可设置首选数据格式。在这一设置，请求的数据串格式可以选择 为 JSON 或 CSV。

```
// 从供应商请求的数据采用 CSV 格式
_chart.DataRequestType = DataRequestType.Csv;
```

注意，要求 JSON 数据串通常要比 CSV 格式更快。如果用户有客户特定的 Rest API 密钥，***DataRequestType 会影响预定数据国内供应商*** AlphaVantage.co 提供数据。

18.4.2 从文件读取

TradingChart 可以通过 ***GetOhlcDataFromFile()***方法直接从.csv 文件读取交易数据。以文件名（和路径）为参数，返回一组 OHLC 数据。

```
// 从文件读取交易数据
OhlcData[] dataFromFile = _chart.GetOhlcDataFromFile("fileName.csv");
```

GetOhlcDataFromFile() 方法会假定文件中的数据排列顺序如下：DateTime、Open、Close、High、Low、Volume (可选)、OpenInterest (可选). DateTime 字段不一定包含时、分和秒。目前，可接受的数据格式为“yyyy-mm-dd”、 “yy-mm-dd” 和 “dd-mm-yyyy”。可接受的分隔符包括连接符(-) 和反斜杠 (/)。

用 ***SetData()*** 方法可以向图表添加已加载的交易数据。

```
// 向图表填设数据
_chart.SetData(dataFromFile, "Symbol", "Data set title");
```

若从文件加载数据，则会自动根据数据集的第一个及最后的 DateTime 值调整图表的时间范围。

18.4.3 自定义数据供应商

除非另行设置，否则 TradingChart 使用预定义的数据提供者。 MarketStack和AlphaVantage.co当前可用。 但是，建议仅将这些数据提供程序用于测试和学习目的，因为所有用户都使用它们，并且通常对数据数量请求有所有限制。 因此，我们建议为最终产品使用自己的ApiKey或数据提供程序。

如果用户想使用一些预定义的数据提供者但使用自己的 API 密钥，应使用SetRestApiKey() -方法。 该方法中，TradingChart在请求交易数据时使用给定的密钥。

```
//从数据提供者发出使用给定密钥的请求.
chart1.SetRestApiKey("apiKey");
```

如果不采用预定义的数据供应商，还有一个可以添加数据的方法，即定义一个独立的 OhlcData 数组，并使用之前所示的 ***SetData()*** 方法。在这种情况下，OhlcData 数组不需要通过从文件中读取数据来填充，而是完全由用户决定从何处以及如何获得数据，然后实现必要的逻辑。

```
OhlcData[] dataArray = new OhlcData[dataPointCount];

// 用获取的数据填充Ohlc-data数组.

_chart.SetData(dataArray, "currency", "dataSetTitle");
```

如果用户想要使用 AlphaVantage.co 供应商，但同时使用独立的 API 密钥，则须使用 ***SetRestApiKey()*** 方法。通过这一方法，可以让 TradingChart 在向 AlphaVantage.co 请求交易数据时采用规定的密钥。

```
// 向AlphaVantage.co请求采用规定的密钥.
chart1.SetRestApiKey("apiKey");
```

请注意，设置 DataProvider 会重置当前的 API 密钥。 因此，仅在设置提供程序之后再调用 SetRestApiKey（）。

使用 Trader 的源代码，用户还可以选择创建自己的数据提供程序类，在这种情况下，搜索栏也会起作用。 如果不应使用预定义的数据提供者，则应将 DataProvider -property 设置为 UserDefined。 这可以防止 TradingChart 尝试自动从提供者获取数据。

```
// Set chart not to use pre-defined data provider.
_chart.DataProvider = DataProvider.UserDefined;
```

不使用预定义数据提供程序时添加交易数据的一种方法是定义自己的 OhlcData -array 并使用 SetData（）-方法。 不必通过从文件中读取数据来填充 OhlcData -array。 完全由用户决定从何处以及如何获取数据，然后实现必要的逻辑。

```
OhlcData[] dataArray = new OhlcData[dataPointCount];

// Fill the Ohlc-data array with the fetched data.

_chart.SetData(dataArray, "currency", "dataSetTitle");
```

18.4.4 调整时间范围

通过图表右下角的按键可以调整图表的时间范围。点击按键可以根据所选的时间值自动修改交易数据。如果数据是从供应商获取的，则将请求从供应商获取的新数据集。

如果所选的时间范围要比可用的数据时长更长，则会根据数据来调整数据范围。比如，如果选择了三年，但是可用的数据时间只有一年，则时间范围显示为一年。另外，所选的世间范围总是根据所加载数据集的最新时间值显示。如果交易数据是从某供应商获取的，最新时间值通常即为当前时间。不过，如果数据是从文件中加载的，情况通常就不是这样了。比如，如果数据的最新日期是31.08.2019，选择一年的时间范围会显示 01.09.2018 – 31.08.2019 之间的数据。

TradingCharts 还有方法可以通过代码来设置自定义世间范围。通过 ***OpenSymbol()*** 方法可以规定起始与结束的时间参数，这时，只会向供应商请求时间范围内的数据。

```
// 请求2018年上半年的数据
_chart.Symbol = "GOOG";
_chart.OpenSymbol(new DateTime(2018, 1, 1), new DateTime(2018, 6, 30));
```

如果没有在 ***OpenSymbol()***规定参数，通常采用当前时间范围。

另外还可以通过 ***SetTimeRange()*** 方法调整时间范围，该方法也可用于从文件加载的数据。

```
// 调整世间范围，显示 2018 年上半年.
chart1.SetTimeRange(new DateTime(2018, 1, 1), new DateTime(2018, 6, 30));
```

如果当前已从供应商加载交易数据，则将根据规定的时间范围和当前 ***Symbol*** 设置自动向供应商请求新的数据集。即使没有数据加载到图表中，***SetTimeRange()*** 也可以运行。除非再次修改，否则调用此方法后的数据请求也使用该时间范围。

注意，点击图标下方的世间范围按键，或者从文件加载数据，都会覆盖通过 ***SetTimeRange()***设置的时间范围。

18.5 数据游标

TradingChart 有内置的数据游标，可以自动追踪可见的交易数据和技术指标，并在一个图例框中显示当前值。游标会追踪鼠标当前所在的图段中的数据和指标。例如，当 Volume 值显示在图表下方

的其他图段中时，就不会追踪 Volume 值，除非将鼠标移动到图表中的图段上。数据游标不会追踪绘图工具。

默认，数据游标会追踪 OHLC 数据的 Close 值。而通过 ***OhlcField*** 属性可以进行更改。

```
// 设置游标追High踪值。
_chart.DataCursor.OhlcField = PriceChartOhlcField.High;
```

图 18-6. 数据游标追踪交易数据以及所有添加的指标。在游标旁的图例框中显示有价格值。由于 Volume 在其他图段，所以不会被追踪。

默认情况下，所有数据和指标都会被追踪。不过，尽可能控制要追踪的系列。可以禁用 ***TrackOhlcData*** 属性，来阻止游标追踪 OHLC 数据。

```
// 数据游标不追踪加载到图表的 OHLC 数据
_chart.TrackOhlcData = false;
```

禁用 ***TrackOhlcData*** 只影响 OHLC 数据，但仍然会追踪其他指标。要阻止游标追踪特定的指标，可以禁用其 ***TrackIndicator*** 属性。

```
// Stops tracking this indicator
simpleMovingAverage.TrackIndicator = false;
```

TrackIndicator 可用于与 OHLC 数据在同样图段绘制的指标。换言之，例如鼠标移动到 Volume 和 Relative Strength Index 等图段指标所属的图段上，这些指标就会一直被追踪。

18.6 数据打包

TradingChart 具有 DataPacking 属性，当启用该属性时，会导致彼此接近的数据值被打包到单个渲染项中。尤其是对于较大的数据集提高了性能，但数据可能不如没有打包的情况下准确。

```
   // Enabling data packing.
_chart.DataPacking = true;
```

18.7 技术指标

TradingChart 有几个内置的技术指标，会根据加载的交易数据进行自动计算。一些指标还可以进行用户定义属性设置，也可以影响到计算结果，例如用于确定指标所依据天数的时间段计数。

18.7.1 添加指标

TradingChart 的 ***Indicators*** 列表，可用于存储所有技术指标。要向图表添加技术指标，首先创建并配置指标，然后将其添加到 ***Indicators*** 列表。

例如，添加 ***RelativeStrengthIndex***:

```
RelativeStrengthIndex rsi = new RelativeStrengthIndex()
{
    LineWidth = 1f,
    PeriodCount = 14,
    HighColor = Colors.Lime,
    LowColor = Colors.Red
};
_chart.Indicators.Add(rsi);
```

18.7.2 删除指标

从 TradingChart 的 ***Indicators*** 列表删除即可删除技术指标。从列表中删除后，指标将会自动释放。

```
_chart.Indicators.Remove(indicator);
```

删除同样类型的所有指标：

```
List<Indicator> itemsToRemove = new List<Indicator>();
foreach (Indicator ind in _chart.Indicators)
{
    if (ind is RelativeStrengthIndex)
        itemsToRemove.Add(ind);
}
foreach (Indicator i in itemsToRemove)
    _chart.Indicators.Remove(i);
```

18.7.3 指标类型和属性

技术指标要么绘制在主要部分（覆盖）的交易数据之上，要么绘制在图表下方的单独部分（研究）中，在这种情况下，该部分是在添加指标时自动创建的。

指标有几个属性，可用于修改它们的外观或它们所基于的计算。 这些属性可以在指标创建时或创建后的任何时间进行修改。

```
// Modifying indicator properties after its creation.
rsi.PeriodCount = 10;
rsi.LineColor = Colors.Blue;
rsi.LineWidth = 2;
```

一般指标属性：

PeriodCount 设置用于计算指标值的时间段数。 默认值取决于指标。

LineWidth 设置适用的指标线的宽度。 如果指标由多行组成，则该属性会影响所有行。

Color 设置适用的指标线的颜色。 如果指标由多条线组成，则每条线都有一个单独的颜色属性。

TrackIndicator 启用后，数据光标将跟踪该指标并显示其当前值在图例框中。 仅适用于显示在交易数据之上的指标。 始终跟踪单独段中的指标。

LimitYToStackSegment 如果启用，指示器将被剪裁到其段之外。

一些技术指标还具有特定于该指标的属性，例如布林带的 **NumberOfStandardDeviations**。

18.7.4 可用指标列表

Envelopes

- Bollinger Band
- Donchian Channels
- Fractal Chaos Bands
- High Low Bands
- Keltner Channels
- Moving Average Envelopes (MAE)
- Prime Number Bands
- Standard Error Bands

- Stoller Average Range Channel

Moving Averages

- Exponential Moving Average (EMA)
- Simple Moving Average (SMA)
- Triangular Moving Average (TMA)
- Time Series Moving Average (TSMA)
- Variable Index Dynamic Average (VIDYA)
- Variable Moving Average (VMA)
- Volume Weighted Moving Average (VWMA)
- Weighted Moving Average (WMA)
- Welles Wilder's Smoothing Average (WWS)

Oscillators – Money Flow

- Accumulation/Distribution (A/D)
- Chaikin Money Flow
- Chaikin Oscillator
- Ease of Movement
- Elder's Force Index
- Klinger Volume Oscillator (KVO)
- Market Facilitation Index
- Money Flow Index
- Negative Volume Index
- On-Balance Volume (OBV)
- Positive Volume Index
- Price Volume Trend
- Trade Volume Index
- Twiggs Money Flow
- Volume
- Volume Oscillator
- Volume Rate of Change (Volume ROC)
- Williams Accumulation Distribution (Williams AD)
- Williams Variable Accumulation Distribution (WVAD)

Oscillators – Price

- Aroon Oscillator
- Awesome Oscillator
- Balance of Power
- Commodity Channel Index (CCI)
- Center of Gravity
- Chande Forecast Oscillator (CFO)
- Chande Momentum Oscillator (CMO)
- Coppock Curve
- Detrended Price Oscillator
- Elder-Ray Index

- Elder Thermometer (custom version)
- Fractal Chaos Oscillator (FCO)
- Intraday Momentum Index (IMI)
- Moving Average Convergence Divergence (MACD)
- Moving Average Convergence Divergence Custom (MACD Custom)
- Momentum Oscillator
- Percentage Price Oscillator (PPO)
- Performance Index
- Pretty Good Oscillator
- Prime Number Oscillator (PNO)
- QStick
- Rainbow Oscillator
- Rate of Change (ROC)
- Relative Strength Index (RSI)
- Stochastic Momentum Index (SMI)
- Stochastic Oscillator
- Stochastic Oscillator Smoothed
- True Strength Index (TSI)
- Ultimate Oscillator
- Ultimate Oscillator Smoothed (UO ST)
- Williams Percent Range (Williams %R)

Statistics

- Correlation Coefficient
- Kurtosis
- Median Price
- Skewness
- Standard Deviation
- Standard Error
- Typical Price
- Weighted Close

Trend Indicators

- Accumulative Swing Index (ASI)
- Average Directional Index (ADX)
- Aroon
- Gopalakrishnan Range Index (GAPO)
- Ichimoku Cloud
- Linear Regression
- Parabolic Stop-and-Reverse (PSAR)
- Random Walk Index
- Range Action Verification Index (RAVI)
- Schaff Trend Cycle (STC)
- Schaff Trend Cycle Signal (STC Signal)
- System Quality Number Trend (SQN Trend)
- Supertrend

- Swing Index
- Triple Exponential Average (TRIX)
- Vertical Horizontal Filter (VHF)

Volatility

- Average True Range (ATR)
- Chaikin Volatility
- Ehler Fisher Transform
- High Minus Low
- Historical Volatility
- Mass Index
- Z-Value

其它指标

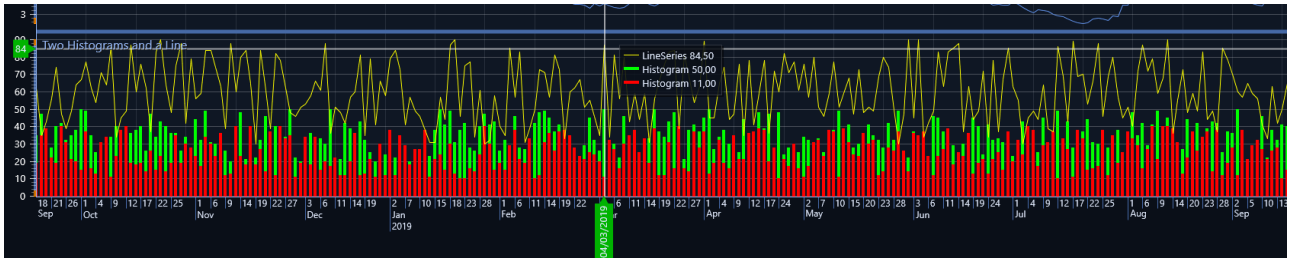
- Open Interest

两个直方图和直线

两个直方图和直线是一个自定义指标，它允许使用两个直方图和一条直线在单独的段中显示给定数据。 与其他指标不同，该数据不是基于加载的 OHLC 数据集。 相反，可以将单独的数据集分配给指标。

Two Histograms 和 Line 特有的属性和方法：

LineSeriesTitle	设置指标线的标题。
HistogramColor1 / HistogramColor2	设置相应直方图中条形的颜色。
HistogramWidth1 / HistogramWidth2	设置相应直方图中条形的宽度。
HistogramTitle1 / HistogramTitle2	设置相应直方图的标题。
HistogramsStacked	启用时，两个直方图相互堆叠。
AddLine()	将线添加到指标。 需要一个 DateTime 数组、Y 值数组、线宽和颜色作为参数。 请注意，这应该在指标添加到图表的指标集合后调用。
AddHistogram()	向指标添加直方图。 需要一个 DateTime 数组、Y 值数组、条形宽度、颜色和直方图索引作为参数。 最后一个参数接受值 1 和 2。数字表示应添加或更新哪个直方图。 请注意，这应该在指标添加到图表的指标集合后调用。



18.8 绘图工具

绘图工具是可以在 TradingChart 上进行自由绘制的可视化工具。除了 ***FreehandAnnotation*** 之外，所有的绘图工具都以两个控制点为基础。当开始绘制第一个绘图工具时，设置第一个点。在点击鼠标左键结束绘制时，设置第二个点。这些控制点在设置好后也可以用鼠标删除掉。绘图工具在绘制时是实时绘制并更新的，控制点移动的时候也是一样。

图 18-7. 绘制一条趋势线。线两端有两个控制点。

18.8.1 添加绘图工具

通过调用绘图工具的 ***StartDrawing()*** 方法可以进行绘制。调用该方法后，左键单击图表就可以开始绘制，接着就可以设置第一个控制点，再次左键单击图表将停止绘制。

```
// 开始绘制一条黄色Trend line（趋势线）
TrendLine trendLine = new TrendLine();
trendLine.LineColor = Colors.Yellow;
_chart.DrawingTools.Add(trendLine);
trendLine.StartDrawing();
```

TradingChart 还具有一个内置的 Tool 绘图工具菜单，在图表的右上角。通过这个菜单还可以更改颜色主题。打开菜单，选择一个绘图工具，实例化相应的绘图工具，并通过内部调用 ***StartDrawing()***进入绘图模式，接着，左键单击图表将添加第一个控制点。

用后台代码添加绘图工具的操作与添加技术指标类似。TradingChart 的 ***DrawingTools*** 列表可用以存储所有的绘图工具。不过，由于绘图工具是基于两个控制点进行绘制的，所以要通过 ***SetControlPointsAndDraw()***方法来设置这些控制点。

```
// 在后台代码添加一个绘图工具
FibonacciArc fibonacciArc = new FibonacciArc();
fibonacciArc.FillEnabled = true;
_chart.DrawingTools.Add(fibonacciArc);

// 设置控制点
fibonacciArc.SetControlPointsAndDraw(
    new Point(10, 200),
    new Point(20, 300)
    );
```

注意，在设置绘图工具的控制点之前，须将绘图工具添加到 ***DrawingTools*** 集中。另外，目前还不能在后台代码中添加 ***FreehandAnnotations***。

18.8.2 删除绘图工具

通过移动鼠标突显一个控制点，再按下 Delete 键，可以删除任何绘图工具。

通过代码删除绘图工具的操作方法与指标类似（参阅删除指标章节）。用 There is also ***DisposeAllDrawingTools()*** 方法可以删除图表中所有的绘图工具。

```
// 清除并释放所有绘图工具
_chart.DisposeAllDrawingTools();
```

当为图表设置新的交易数据，或通过世间范围按键更改时间范围时，默认所有的绘图工具会自动清除。通过 ***AutoClearDrawingTools*** 属性可以控制这一功能。默认值为 ***NewDataAndTimeRange***，即可引至上述操作。将该属性设置为 ***NewData***，则只会在加载新数据集时清除绘图工具，而在更改时间范围时则不会清除。设置为 ***None*** 时，则组织一切自动清除（调用 ***DisposeAllDrawingTools()***方法仍然有效）

```
// Clear drawing tools when new data set is loaded but not when time range is changed.
_chart.AutoClearDrawingTools = ClearDrawingTools.NewData;
```

18.8.3 绘图工具的类型

所有绘图工具都有几个属性。

常用绘图工具属性：

LineColor	设置绘图工具所有线条的颜色。 也会影响控制点。
LineWidth	设置绘图工具所有线条的宽度。 也会影响控制点。
LabelColor	如果绘图工具有文本标签，改变它们的颜色。
Magnetic	启用后，绘图工具线会自动附加到垂直方向上最近的 OHLC 值。 默认情况下禁用。
LimitYToStackSegment	如果启用，绘图工具将被剪裁到主线段之外。

Elliot Wave 艾略特波浪

艾略特波浪在几个控制点之间绘制波浪图案。

艾略特波浪特有的属性：

WaveType	波型的长度和使用的标记不同。

图 18 8. 艾略特波浪已绘制到图表上

Fibonacci Arc 斐波那契弧线

在两个控制点之间绘制了一条趋势线，随后是多条弧线与该线在 38.2%、50.0%、61.8% 和 100% 的水平相交。 圆弧以第二个控制点为中心。

属性：

FillEnabled	启用时，圆弧之间的区域是彩色的。
LabelDistance	控制标签与其各自弧线的距离（以像素为单位）。
FullCircle	启用时，圆弧被绘制为完整的圆。

图 18 9. 斐波那契弧线已添加到图表中 c

Fibonacci Fan 斐波那契扇形

在两个控制点间绘制一条趋势线，，然后从第一个点开始画几条 Fibonacci Fan（斐波那契扇形）线，并依据在 38.2%、50.0% 和 61.8%处的斐波纳契回撤位，在第二个点的 X 值处与一条“invisible（隐形的）”垂线相交。

属性：

FillEnabled – 启用后，给扇形线内的区域着色。

LabelDistance – 控制标签与各自扇形线之间的像素距离。

图 18-10. A Fibonacci Fan with colored fills enabled (*FillEnabled = true*).

Fibonacci Retracements （斐波纳契回调线）

在两个控制点间绘制一条趋势线，，然后根据趋势线的所选价格范围（高度）画几条水平回调线。回调线在斐波纳契回撤位 38.2%、50.0% 和 61.8%处绘制。

属性：

FillEnabled – 启用后，回调线间的区域将着色。

LabelDistance –控制标签与各自回调线之间的像素距离。

LimitYToStackSegment – 启用后， 斐波纳契回调线在主图段外的将剪切掉。

图 18-11. 填充颜色了的 Fibonacci Retracements 线 (*FillEnabled = true*).

Freehand Annotation 手绘标注

基于鼠标移动绘制任意形状的多边形。在多边形内可以显示文本。

属性：

FillColor – 改变标注的填充色。

BorderColor – 改变标注的边框颜色。

BorderWidth – 改变标注的边框宽度。

FillEnabled – 启用后，用通过 ***FillColor*** 设置的颜色填充标注。

ShowText – 启用后，在标注内显示文本。文本的位置会自动计算为处于重心位置。

AnnotationText –启用 ***ShowText*** 后，设置为显示文本。

TextColor –设置文字颜色。

FontSize – 设置文字字体大小。

图 18-12. 绘制的手绘标注。在标注中间显示文本。(*ShowText = true*).

Head and Shoulders 头肩形态

头肩形态绘制了具有三个峰值的基线。

属性：

FillColor 更改峰区域的填充颜色

图 18-8. 交易图表中的头肩形态

Linear Regression 线性回归

在两个控制点间绘制一条线性回归线。接着，根据所选通道类型，绘制两条轨道线，一条在回归线之上，一条在回归线之下。

属性：

LineColor – 更改回归线以及轨道线的颜色。也会影响延长线。

LineWidth – 更改回归线以及轨道线的宽度。也会影响延长线。

ShowExtensions – 启用后，绘制虚延长线，延长至所加载交易数据的最后一个值。

FillEnabled – 启用后，轨道线与回归线之间的区域将着色。

ChannelType –仅确定所使用的线性回归轨道（Linear Regression Channel）类型，如标准偏差通道、Raff 通道和回归线。

NumberOfStandardDeviations –设置定义通道线与回归线之间距离的标准差数。仅当 ***ChannelType*** 设置为 ***StandardDeviations*** 时生效。

LimitYToStackSegment – 启用后，线性回归轨道在主图段外的将前切掉。

图 18-9. 基于标准偏差的线性回归通道（Linear Regression Channel ）, *ChannelType = StandardDeviations, NumberOfStandardDeviations = 1.*

Pitchfork 安德鲁斯干草叉

安德鲁斯干草叉，可用于确定股票价格的支撑位和阻力位。 它在图表上放置三个控制点，并从第一个点到其他点的中点画一条线。

属性：

FillColor 更改干草叉线之间区域的填充颜色.

Figure 18-10. 更改干草叉线之间区域的填充颜色.

Trend Line 趋势线

在两个控制点之间绘制一条直线。

属性：

ShowExtensions 启用后，绘制延伸至加载交易数据的第一个和最后一个值的虚线延伸线

图 18-11. 启用扩展的趋势线 (ShowExtension = true)。

Triangle 三角形

基于三个控制点绘制三角形。

属性：

FillColor 更改三角形的填充颜色

图 18-12. 交易图表上的三角形.

XABCD Pattern XABCD 模式

在图表上绘制五点形态。

属性：

FillColor 更改图案的 XAB 和 BCD 区域的颜色.

ShowRatios 启用后，显示不同腿之间的比率值。 默认设置为 true。

图 18-13. XABCD 模式已添加到图表中。 ShowRatios 已启用

其它可用绘图工具

- Arrow
- Ellipse
- Fibonacci Time Zones
- Horizontal Line
- Horizontal Ray
- Plain text
- Rectangle
- Text Box
- Vertical Line

18.9 TradingChart 故障排除

当 TradingChart 没有如期运行，会自动显示一些错误信息。信息会在图表顶部以及 Visual Studio 的输出窗口显示。当单击图表或将新数据集加载到图表时，图表上方的错误信息将删除。

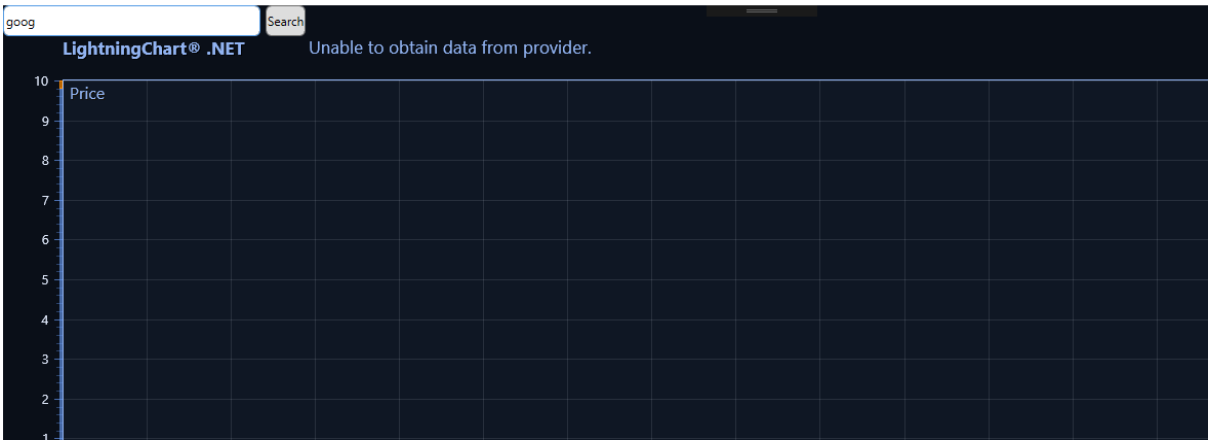


图 18-14. 图表上方显示出一条错误信息。

通过 ***ShowErrorMessages*** 属性可以禁用图表上方所示的错误消息，这时，错误信息只会在输出窗口中显示。

```
// 不在图表上方显示错误信息。
_chart1.ShowErrorMessages = false;
```

18.9.1 错误列表

下面列出了几条错误信息、可能的原因，以及如何修复。注意，所列内容并不包含所有可能发生的错误。如果某个错误无法解决，请发邮件至 support@lightningchart.com 与 Arction 的技术支持联系。

信息：*“Unable to obtain data from provider.”（无法从供应商处获取数据）*

说明： TradingChart 无法连接到数据供应商，因此无法获取数据。

可能有效的解决办法：确保互联网连接正常。

信息：*“No data for given time range.”*（制定的时间范围内没有数据）

说明： 在指定的时间范围内没有发现交易数据。当从供应商处加载数据集或从文件处加载数据集，都可能出现此信息。

可能有效的解决办法：尝试使用其他时间范围。

信息：*“Time range set to: start time – end time”*（时间范围设置为：起始时间——结束时间）

说明：这并不是一个实际的错误。相反，这通知的是时间范围已更改，但更改不可见，因为没有数据加载到图表中。

可能有效的解决办法： 在加载了数据集后调用 ***SetTimeRange()*** 。或者，只需从供应商加载一个新的数据集，这时，将使用之前设置的时间范围。

信息：*“End time should be after start time.”*（结束时间应在起始时间之后）

说明： 手动设置世间范围时，未能正确设置。

可能有效的解决办法： 确保时间范围的起始时间在结束时间之前。

信息：*“Couldn’t obtain Symbol information.”*（无法获取 *Symbol* 信息）

说明：当用 ***OpenSymbol()***方法调用时发生。该方法会尝试获取相应的 symbol 信息以及数据集。该信息表示获取 symbol 信息失败。而某些时候数据集依然可以加载。

可能有效的解决办法：检查给定的 symbol 字符串是否正确。如果错误仍然发生，数据供应商本身可能有一些临时问题。

信息：*“The requested stock could not be found.”*（找不到请求股票）

说明： 数据供应商无法找到所给定的 symbol 或证券名称。当用代码调用 ***OpenSymbol()***方法或使用搜索栏时会发生此错误。

可能有效的解决办法：检查给定的 symbol 字符串是否正确。如果错误仍然发生，数据供应商本身可能有一些临时问题，或者仅是无法获取该特定证券数据。

信息：*“Invalid API call. Unable to fetch data with the given parameters.*无效 API 调用。无法使用给定参数获取数据。

说明：数据供应商无法用给定参数找到数据。通常在用代码调用 ***OpenSymbol()***方法时发生该错误。

可能有效的解决办法：检查给定的 symbol 字符串是否正确。如果错误仍然发生，数据供应商本身可能有一些临时问题，或者仅是无法获取该特定证券数据。或者，如果使用的时间范围为几年，该证券可能没有每周或每月可用的数据。

信息：*“Call limit reached.”*（调用达到限制）

说明：数据供应商可能有调用限制，例如每分钟或每天有最大数据请求量。 该错误表示已达到限制。

可能有效的解决办法：稍后再尝试请求数据。或者，获取一个允许更多数据请求的个人 Rest API 密钥。

信息：*“Invalid data provider API key.”*（无效数据供应商 API 密钥）

说明：当使用客户特定的密钥时，会给 ***SetRestApiKey()***方法一个不正确的 Rest API 密钥。

可能有效的解决办法： 确保使用有效密钥。如果在使用预定的数据供应商时发生该错误，请尝试在代码中将 DataProvider 设置为 MarketStack（或 AlphaVantage），可重置连接。

```
_chart.DataProvider = DataProvider.MarketStack;
```

信息：*“Data array contains no points.”*（数据数组不包含点）

说明：仅当用代码调用 ***SetData()***方法，同时给它一个空的数据数组时，才会出现此消息。

可能有效的解决办法：确保给 ***SetData()***的 ***OhlcData*** 数组不是无效或空的。

信息：*“Data contains empty values.”*（数据包含空值）

说明： 这可以理解为一个警告，而不是实际的错误信息。这说明加载的 OHLC 数据集包含至少一个空值，Open、High、Low、Close 或 Volume 字段的字符串值为“”。注意，0 不被视为空值，而且将在图表上绘制。

可能有效的解决办法：这一信息并不妨碍 TradingChart 运行。数据将正常加载并绘制，而空值将跳过。如果需要，由用户来实现处理这些情况的逻辑。

信息：*“Unable to use pre-defined data provider.”*（无法使用预定的数据供应商）

说明： TradingChart 因连接问题，无法使用预定的数据供应商。

可能有效的解决办法：确保网络正常连接。如果这一问题仍然出现，请联系的 Arction 技术支持。

信息：*“Problem with connection to Arction server.”*（与 Arction 服务器连接有问题）

说明： TradingChart 无法从 Arction 的服务器获取所需信息以使用预定的数据供应商。

可能有效的解决办法：此错误很可能不是由用户引起的。因此，唯一的解决方法是使用另一个数据供应商。 或者，联系 Arction 的技术支持，以获取关于当前服务器状态的更多信息。

18.9.2 常见问题解答

*-如何防止不断发生“Call limit reached error”（错误：*调用达到限制*）?*

-该错误通常在使用预定的数据供应商 AlphaVantage.co 时发生。所有测试 TradingChart 的用户都使用 Arction 独立的 Rest API 密钥，同时由于该密钥有调用限制（每分钟调用），因此在使用率高时可能会发生此错误。所以， ***Arction 建议用户仅适用预定的供应商进行测试，***并获得独立的 API 密钥或使用其他数据供应商，以确保在构建自己的应用程序时，可以在任何时候都能正常获取数据。

-从供应商获取数据时感觉比较慢，能做些什么变快点吗?

-这主要取决于请求的数据集有多大以及数据供应商本身。如果使用预定的数据供应商 AlphaVantage.co，可能会因为请求的数据非常大而发生这一问题。Currently, 目前，AlphaVantage 仅支持的输出大小选择有“compact”型 （最后 100 个数据点）和 “full” 型（全部数据）。所以，请求的数据集再小一点可能会有所帮助。另外，获取 json 格式的数据通常要比获取 csv 格式的数据要快。

注意，TradingChart 会自动缓存获取的数据，所以大多数情况下，更改时间范围并不会引发图表向供应商请求新数据。因此，首次请求时会比之后的请求花费的时间更久一些。

19. SignalGenerator 组件

演示示例： Areas; Oscilloscope; SignalGenerator -> speakers; Intensity persistent layer, signal; High-speed data, stacked axes （WinForms only）

SignalGenerator 组件可以用来生成实时信号。该信号是由不同波形叠加而成的。多个 ***SignalGenerator*** 组件可以通过主从关系进行连接，以产生同步的多信道输出。 在用 LightningChart 开发信号监测或数据采集软件时，***SignalGenerator*** 非常有用。

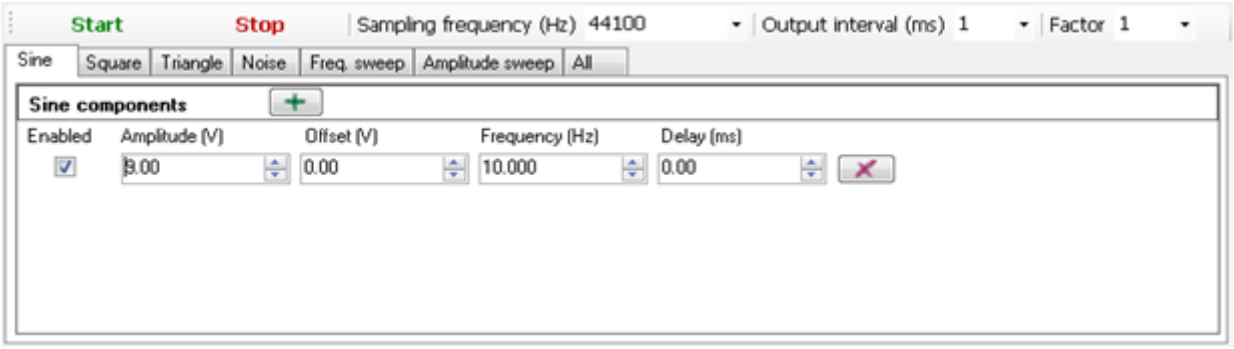


图 199-1. 选择信号发生器组件的正弦页面。

波形分为以下几类： ***Sine***、***Square***、***Triangle***、 ***Noise***、 ***Frequency sweep*** 和 ***Amplitude sweep***。在组件中可以看到它们各自的选项卡页面。在正弦页面，可以添加正弦波形。在 Sine 页面，可以添加正弦波形；在 Square 和 Triangle 页面，分别可以添加正波形和三角波形；在 Noise 页面，可以随机生成噪音波形；在 Frequency 和 Amplitude sweep 页面，可以添加频率和振幅扫描；在 ***All*** 页面，可以以堆叠视图设置所有波形。

19.1 样频率、输出间隔和系数

通过 ***SamplingFrequency*** 可知道每秒产生多少个信号点。更高的采样频率可以产生更准确的信号，但同时也增加了数据流和额外负担。通过高采样频率，可以得到含有高频率的信号。采样频率必须大于最大信号频率的两倍才能满足奈奎斯特（Nyquist）采样定理。

OutputInterval 可设置计算的输出样本的首选间隔，以毫秒为单位。例如，如果 ***OutputInterval*** 设置为 100，那么在每 100 ms 周期之后，每秒将收到 10 次采样包。使用较低的值将带来更顺畅的实时监控输出。注意，***OutputInterval*** 并不准确，可能会随着计算机负载的变化而变化。如果周期比预期的长，输出数据流将自动生成更多的采样。数据流也将在高数据速率和繁重的计算机开销下成形。

Factor 可根据所选值增加输出采样。例如要生成 mV 信号而不是 V 信号，可设置 ***Factor*** 为 1E-3。

19.2 正弦波形

利用 ***Amplitude***、***Offset***、***Frequency*** 和 ***DelayMs*** 参数构造正弦波形。 ***Amplitude*** 是与零电平的最大电压差。注意，其总范围是双相的。峰峰值将是 2 * ***Amplitude***。***Offset*** 是添加到信号上的直流电平。换句话说，在值域范围内，正值使信号向上移动，负值使信号向下移动。***Frequency*** 以赫兹表示信号周期数。每秒一个周期的频率是 1 赫兹。***DelayMs*** 信号的延迟以毫秒为单位。

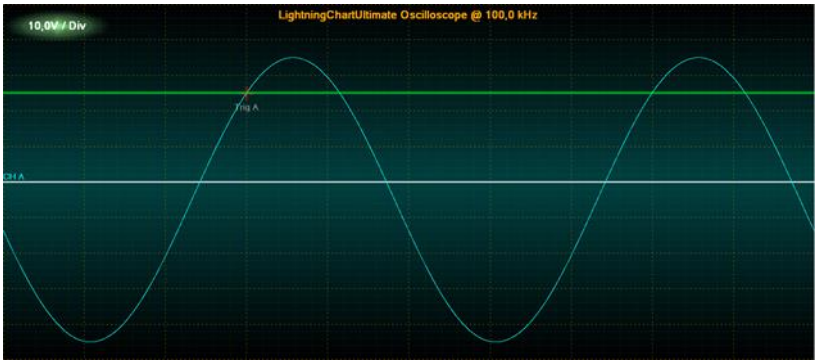


图 199-2. 采用如上设置的一种简单的正弦波形信号

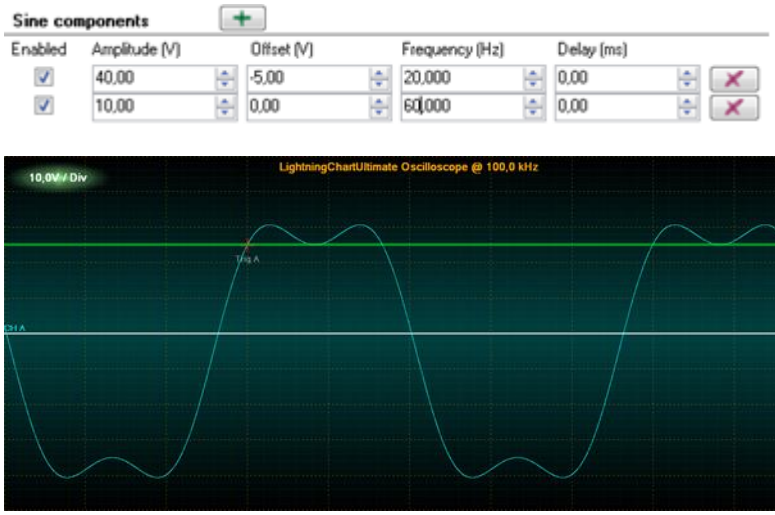


图 199-3.采用如上设置的二个正弦波形信号

19.3 方波形

方形波比正弦波形多一个参数——***Symmetry***。***Symmetry*** 的范围为 0...1。通过 ***Symmetry*** 可以知道信号处于高处状态的时间，且与循环周期相关。值为 0.5 时，信号在高处和低处的长度相等。

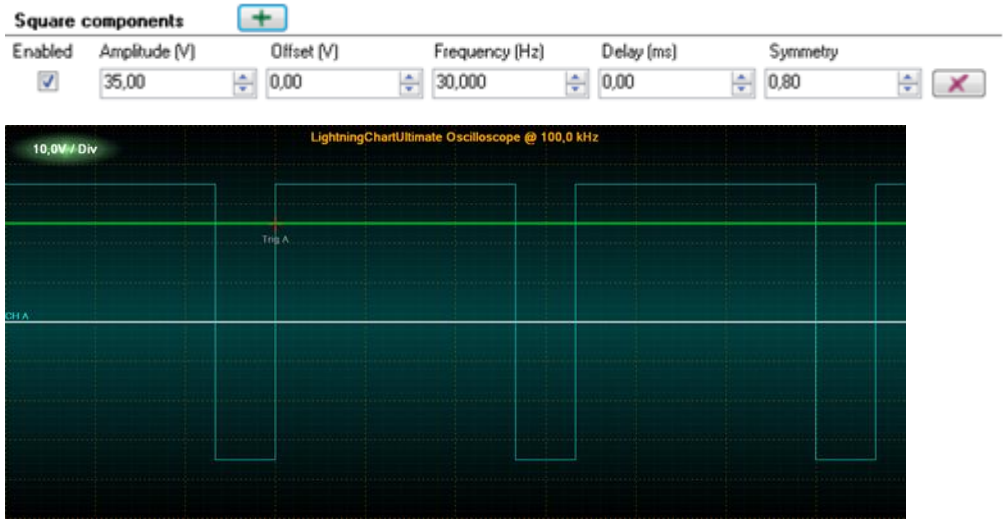


图 199-4. 一个方波形信号，对称度为 0.8。设置如上。

19.4 三角波形

三角形波形也有 ***Symmetry*** 参数。它控制三角形倾斜的方式。0.5 是对称三角形的值。小于 0.5 的值向左倾斜，大于 0.5 的值向右倾斜。

图 199-5. 一个三角波形信号，对称度为 0.7

19.5 噪音波形

噪音波形是随机产生的信号。点随机出现在***–Amplitude*** 和***+Amplitude*** 之间。

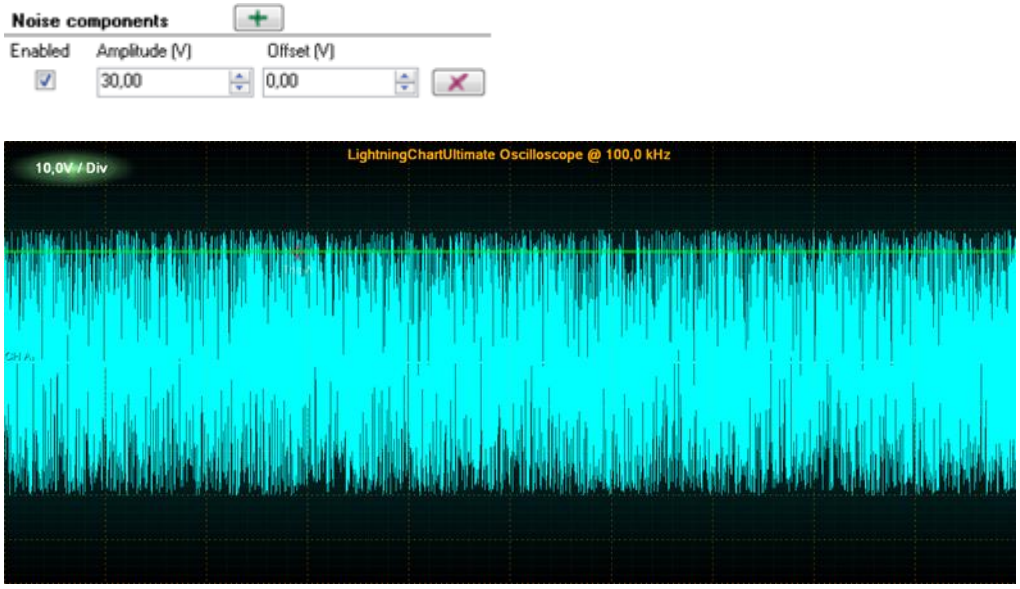


图 199-6.产生随机数据点在振幅-30 和+30 之间的噪音波形信号。

19.6 频率扫描

频率正弦扫描在给定的时间周期内以恒定振幅从频率 1 持续到频率 2。用 ***FrequencyFrom*** 设定起始频率，用 ***FrequencyTo*** 设定结束频率，用 ***Amplitude*** 设定恒定振幅，用 ***DurationMs*** 以毫秒为单位设定持续时间。

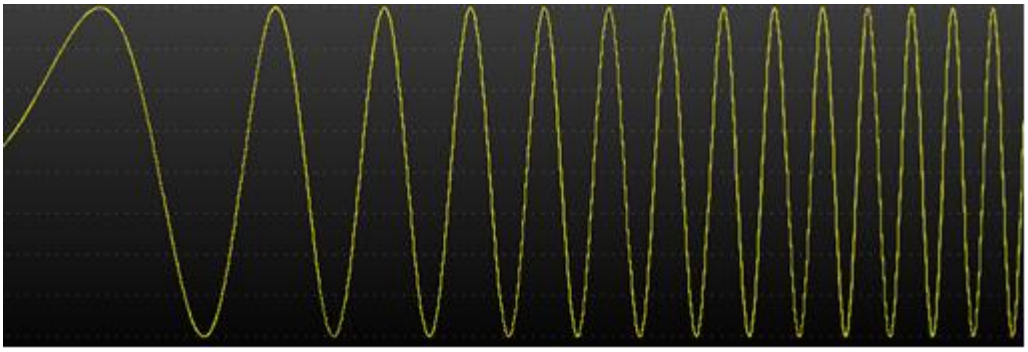


图 199-7..频率扫描

19.7 振幅扫描

振幅正弦扫描在给定的时间周期内从振幅 1 到振幅 2，频率不变。用 ***AmplitudeFrom*** 可设置起始振幅，用 ***AmplitudeTo*** 可设置结束振幅，用 ***Frequency*** 可设置恒定频率，用 ***DurationMs*** 可以毫秒为单位设置持续时间。

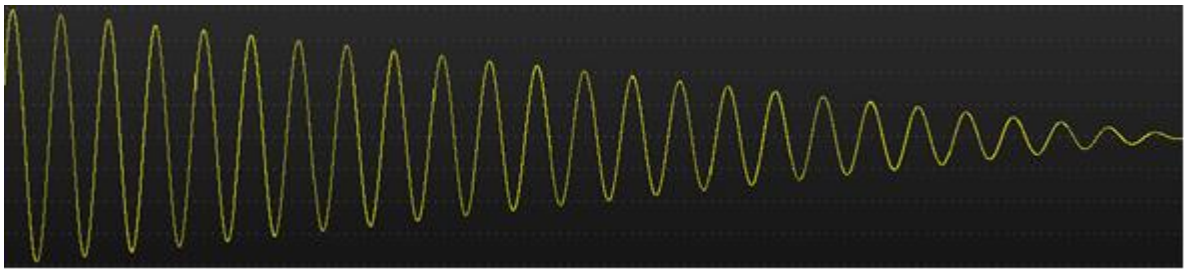


图 199-8.振幅扫描

19.8 开始与停止

通过按下 Start 按钮或调用 ***Start*** 方法来启动生成器。通过按下 Stop 按钮或调用 ***StopRequest*** 方法来停止生成器。***Stopped*** 事件将在停止完成时触发。

19.9 采用主从结构的多信道生成器

多个 ***SignalGenerator*** 组件可以连接起来，生成同步的多信道输出。

MasterGenerator 控制所有生成器的采样频率、启动、停止和输出。它在输出数据流中产生第一个信道。

通过设定从生成器和主生成器的 ***MasterGenerator*** 属性，可以将二者连接起来。信号波形可以自由定义。从生成器由主生成器启动和停止。它们按照连接顺序获取输出数据流信道索引。在启动主生成器之前，必须连接从生成器。

19.10 输出数据流

输出数据流由二维数组组成，通过 ***DataGenerated*** 事件处理程序获得。通常，事件在每个 ***OutputInterval*** 之后引发。

事件处理程序获取对采样数组的引用，以及在此间隔期间收到的第一个采样包的时间戳。采样数组的第一个维度表示信道，第二个维度表示每个信道的采样。所有信道的采样数相等。

触发 ***DataGenerated*** 事件:

```
m_signalGenerator.DataGenerated += m_signalGenerator_DataGenerated;

private void m_signalGenerator_DataGenerated(DataGeneratedEventArgs args)
{
   // 事件代码
{
```

为研究数据流的信道数，可获取第一个维度的长度:

```
channelCount = args.Samples.Length;
```

要获得一个信道的采样数:

```
sampleBundleCount = args.Samples[0].Length;
```

下面的代码将演示如何在更新实时监视滚动条位置的同时，将输出数据直接转发到 LightningChart 的 ***SampleDataSeries*** 列表。

```
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);
   }

   //设置最新滚动位置x
   newestX = args.FirstSampleTimeStamp + (double)(sampleBundleCount - 1) /
   generatorSamplingFrequency;
   chart.ViewXY.XAxes[0].ScrollPosition = newestX;
   chart.EndUpdate();
}
```

注意，设置 ***args.Samples[0]*** 时，可以访问主生成器的数据； ***args.Samples[1]*** 则可以访问第一个从属生成器数据； ***args.Samples[2]*** 则是第二个丛书生成器，以此类推。

20. SignalReader 组件

演示示例： Signal reader; Waveform and spectrum; Waveform, 3D spectrogram; Audio L+R, area, spectrogram

SignalReader 组件可以从信号源文件读取数据并以选定的速率回放。 ***SignalReader*** 输出数据流格式与 ***SignalGenerator 类似***（参阅第 **Error! Reference source not found.**章节）.

当前，***SignalReader*** 组件支持 wav 和 sid 格式。

20.1 关键属性

FileName 定义要打开的文件，例如 "c:\\wavedata\\audioclip1.wav"

Factor 设置输出系数。原始信号采样乘以这个值。

OutputInterval 与 ***SignalGenerator*** 的属性类似（参阅第 **Error! Reference source not found.**章节）.

IsLooping 可以让文件读取当到达文件的末尾时跳到文件的开头。

打开文件后，可以使用以下属性获取文件信息:

ChannelCount: 文件信道数。

SamplingFrequency: 以 Hz 为单位的采样频率。

FileSize: 以字节为单位的文件大小 。

Length: 每个信道的采样数。这不可能对所有信号文件格式都精确。

IsReaderEnabled: 状态命令是组件已启动并读取数据。若 ***Looping*** 设置为 false，且到达了文件末尾，那么 ***IsReaderEnabled*** 将变更为 false。

20.2 快速打开文件进行回放

利用一个文件名调用 ***OpenFile（…）*** 方法。该文件名必须具有受支持格式的扩展名。然后，调用 ***Start（）*** 方法。

```
signalReader.OpenFile（"c:\\wavedata\\audioclip1.wav"）;

signalReader.Start（）;
```

接着开始播放一个 PCM 格式的 WAV 文件。

调用 ***StopRequest***（）方法可以停止播放。

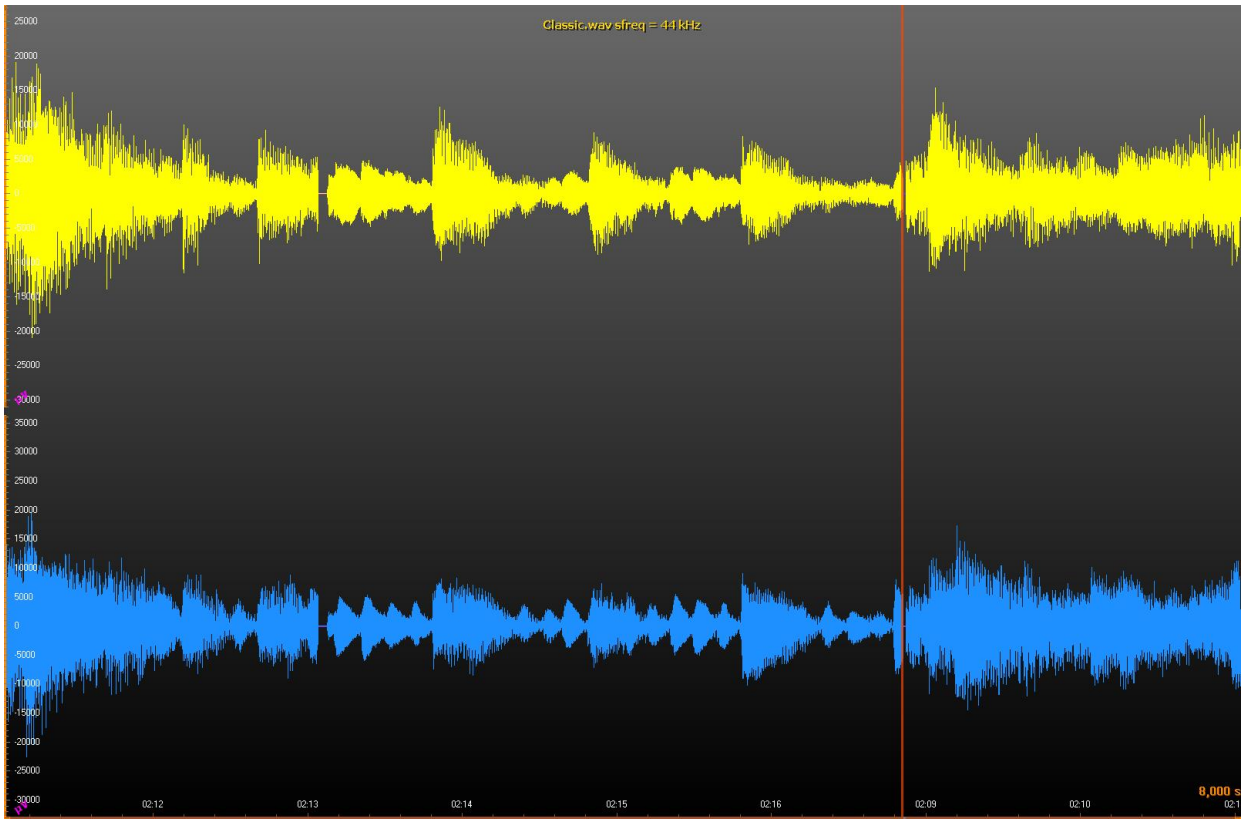


图 20-1. SignalReader 读取一个 wav 文件，然后 LightningChart SampleDataSeries 绘制信号。用一条光标线标记当前读取位置以及 X 轴滚动位置。

21. AudioInput 组件

演示示例： Audio input, waveform; Audio input, spectrogram

AudioInput 组件可以让用户将来自 Windows 录音设备的信号录制为 System.Double 类型值。这些值会在 LightningChart 上渲染，发送到 AudioOutput 组件，存为文件等…

21.1 属性

BitsPerSample – 获取或设置逐帧采样分配多少位。可支持的值为 8 和 16 位。如果要采用其他值，则可用 16 位替代。当设置 ***IsInputEnabled*** 为 ***false*** 时，可以对其设置。

IsInputEnabled – 获取或设置此实例的状态（例如启动或结束）。将此属性设置为 ***true*** 与调用 ***Start*** 方法相同，而设置为 ***false*** 则与调用 ***Stop*** 方法相同。

IsStereo – 获取或设置是使用两个信道（立体声）还是仅使用一个信道（单声道）。当设置 ***IsInputEnabled*** 为 ***false*** 时，可以对其设置。

LicenseKey – 以常规或加密格式获取或设置许可证密钥。

RecordingDevice – 获取或设置当前录音设备。当设置 ***IsInputEnabled*** 为 ***false*** 时，可以对其设置。将此属性设置为 ***null***，则使用 Windows 的默认录音设备。

SamplesPerSecond – 获取或设置采样频率。当设置 ***IsInputEnabled*** 为 ***false*** 时，可以对其设置。

ThreadInvoking – 获取或设置此实例是否自动将其事件同步到主 UI 线程，从而消除在调用方一端调用 ***Control.Invoke*** 方法的需要。

Volume – 获取或设置音量，范围从 0 到 100。当设置 ***IsInputEnabled*** 为 ***false*** 时，可以对其设置。

21.2 Methods （方法）

GetRecordingDevices – 使用此静态方法可获得可用的 Windows 录音设备列表。

RequestStop – 表示该 ***AudioInput*** 实例停止。退出此方法后并不会立即停止。通过订阅 ***Stopped*** 事件，当一切都已停止后，再通知调用方。

Start –开始从选定的录音设备读取音频。***Started*** 事件会在内部线程即将启动时触发。

21.3 Events （事件）

DataGenerated – 当生成一组新的音频数据时才发生。 数据及其第一个采样的时间戳都可以从 ***DataGeneratedEventArgs*** 对象读取，该对象是作为参数提供。

Started –当音频输入开始时发生。***StartedEventArgs*** 对象作为参数提供，包括三个公共字段：***BitsPerSample***、***ChannelCount*** 和 ***SamplesPerSecond***。

Stopped – 当音频输入已停止时发生。

21.4 用法（WinForms）

本章介绍了 ***AudioInput*** 类在 WinForms 版本的用法。关于 WPF 版本的将在第 21.5 章节介绍。

21.4.1 创建

通过代码手动创建一个新的 ***AudioInput*** 实例，或者从 Visual Studio 的工具箱中将它拖放到窗体、用户控件等上。

如果不需要显示 GUI （例如，如果使用自己的 GUI 或从源代码控制 ***AudioInput*** 对象），那可以设置 ***Visible*** 属性为 ***false***。我们常常建议设置 ***Parent*** 属性，以便在释放父控件时自动释放 ***AudioInput*** 实例。如果没有父类，那么当不再需要 ***AudioInput*** 实例时，应该调用 ***Dispose*** 方法。如果通过 Visual Studio 的工具箱创建了一个新的 ***AudioInput*** 实例，那么 ***Parent*** 属性则会自动设置。

建议设置 ***LicenseKey*** 属性，这样，***AudioInput*** 实例会使用一种显式的许可密钥，而不是试图从 Windows 的注册表中找到一个。如果使用的是试用版/许可，***LicenseKey*** 属性可以保留为其默认值。

21.4.2 事件处理

要从 ***AudioInput*** 实例中获取新采样，用户需要至少订阅 ***DataGenerated*** 事件。当触发 ***DataGenerated*** 事件时，来自 ***DataGeneratedEventArgs*** 对象的新采样和第一个采样时间戳会作为参数获得。

订阅 ***Started*** 事件，可以了解 ***AudioInput*** 实例何时启动了其音频采样任务。***StartedEventArgs*** 对象提供与 ***AudioInput*** 有关的信息作为参数，例如逐帧采样位数，是流音频单声道或立体声，以及每秒生成多少采样。

订阅 ***Stopped*** 事件，可以了解 ***AudioInput*** 实例何时停止。该事件没有参数，它的唯一目的是告诉用户什么时候一切都停止了。

21.4.3 配置

设置 ***ThreadInvoking = true***，可以让 ***AudioInput*** 实例自动将其事件同步到主 UI 线程，但是要确保 ***AudioInput*** 实例有一个有效的父控件。

默认情况下，***ThreadInvoking*** 设置为 ***false***，所以如果在事件处理程序中更新 GUI，不要忘了调用 ***Control.Invoke*** 方法。

设置 ***RecordingDevice*** 属性可以使用其他 Windows 的录音设备，而不是默认的。通过采用 ***AudioInput*** 的静态方法 ***GetRecordingDevices***，可以获得所有可用的录音设备。

音量可以通过 ***Volume*** 属性来控制。有效值为从 0 到 100，其中 0 即为静音，100 是最大音量。当 ***AudioInput*** 实例启用时（即生成采用），也可以设置音量。

设置 ***SamplesPerSecond*** 属性以使用不同的采样率，而不是默认的值（44100 Hz）。在 ***AudioInput*** 实例开启后设置此属性无效果。

要使用单声道音频，而不是立体声（默认），可以设置 ***IsStereo*** 为 ***false***。在 ***AudioInput*** 实例开启后设置此属性无效果。

如果逐帧采样是 8 位而不是 16 位（默认值），则将 ***BitsPerSample*** 属性设置为 8。有效值为 8 何 16 （默认值）。此限制来自 PCM 波格式。在 ***AudioInput*** 实例开启后设置此属性无效果。

21.4.4 开始

要开始 ***AudioInput*** 实例，可以设置 ***IsInputEnabled*** 属性为 ***true***，或者调用 ***Start*** 方法。 然后，***DataGenerated*** 事件可提供一组新的音频采样，例如，可以使用 LightningChart 实例渲染。

21.4.5 停止

要停止 ***AudioInput*** 实例，可以设置 ***IsInputEnabled*** 为 ***false***，或调用 ***RequestStop*** 方法。***RequestStop*** 方法不会立即停止。相反，它会向 ***AudioInput*** 实例发送信号，让它尽可能快地停止。订阅 ***Stopped*** 事件可以知道 ***AudioInput*** 实例何时停止了。

21.5 用法（WPF）

本章介绍了 ***AudioInput*** 类在 WPF 版本的用法。***AudioInput*** 在 WPF 版本的用法与在 WinForms 版本中十分类似。但是，有些地方需要 WPF 用户注意。

21.5.1 创建

创建一个新的 AudioInput 实例，可以在代码隐藏中手动创建，也可以通过将其从 Visual Studio 的工具箱拖放到窗口、用户控件等上来创建。

如果不需要显示 GUI（即，如果从源代码控制自己的 GUI 或 AudioInput 对象），则使用 LightningChartLib.WPF.SignalProcessing 命名空间中的 AudioInput。 这个特定的类派生自 FrameworkElement，它的所有属性都是可绑定的。 为方便起见，安装 LightningChart® .NET SDK 后，还可以从 Visual Studio 的工具箱中找到 LightningChartLib.WPF.SignalProcessing.AudioInput，以便将其拖放到窗口、用户控件等中，然后在 XAML 代码中移动到任何位置 是需要的。 必要的 XML 命名空间将通过这种方式自动添加。

还有一个用于 AudioInput 的现成 GUI。 它可以在 LightningChartLib.WPF.SignalProcessing.Gui 命名空间中找到。 安装 LightningChart® .NET SDK 后，Visual Studio 的工具箱也有它。 请注意，这只是 AudioInput 类的 GUI，但它包含可以访问的 AudioInput 类的实例。 换句话说，不需要创建新的单独的 AudioInput 实例。

建议设置 LicenseKey 属性，以便 AudioInput 实例使用显式许可证密钥，而不是尝试从 Windows 注册表中查找许可证密钥。 使用试用版/许可证时，LicenseKey 属性可以保留为其默认值。

22. AudioOutput 组件

演示示例： SignalReader -> speakers; SignalGenerator -> speakers

利用 ***AudioOutput*** 组件，用户可以将 System.Double 信号数据转换成音频流，然后通过扬声器回放或发送到声音设备的音频输出连接器。

22.1 属性

Balance – 获取或设置音频播放平衡。有效值为-100 t 至 100。-100 表示声音只能通过左扬声器播放。0 表示两个扬声器都输出音频。100 表示音频只能通过右扬声器播放。

BitsPerSample – 获取或设置逐帧采样分配多少位。可支持的值为 8 和 16 位。如果要采用其他值，则可用 16 位替代。当设置 ***IsOutputEnabled*** 为 ***false*** 时，可以对其设置。

IsOutputEnabled – 获取或设置此实例的状态（例如启动或结束）。将此属性设置为 ***true*** 与调用 ***Start*** 方法相同，而设置为 ***false*** 则与调用 ***Stop*** 方法相同。

IsStereo – 获取或设置是使用双信道（立体声）还是仅使用一个信道（单声道）。当设置 ***IsOutputEnabled*** 为 ***false*** 时，可以对其设置。

LicenseKey – 以常规或加密格式获取或设置许可证密钥。

PlaybackDevice – 获取或设置当前播放设备。当设置 ***IsOutputEnabled*** 为 ***false*** 时，可以对其设置。将此属性设置为 ***null***，则使用 Windows 的默认播放设备。

SamplesPerSecond – 获取或设置采样频率。当设置 ***IsOutputEnabled*** 为 ***false*** 时，可以对其设置。

Volume – 获取或设置音量 （0-100）。当设置 ***IsOutputEnabled*** 为 ***false*** 时，可以对其设置。

23. SpectrumCalculator 组件

演示示例： Waveform and spectrum

SpectrumCalculator 组件可用以实现在时域和频域之间进行转换。

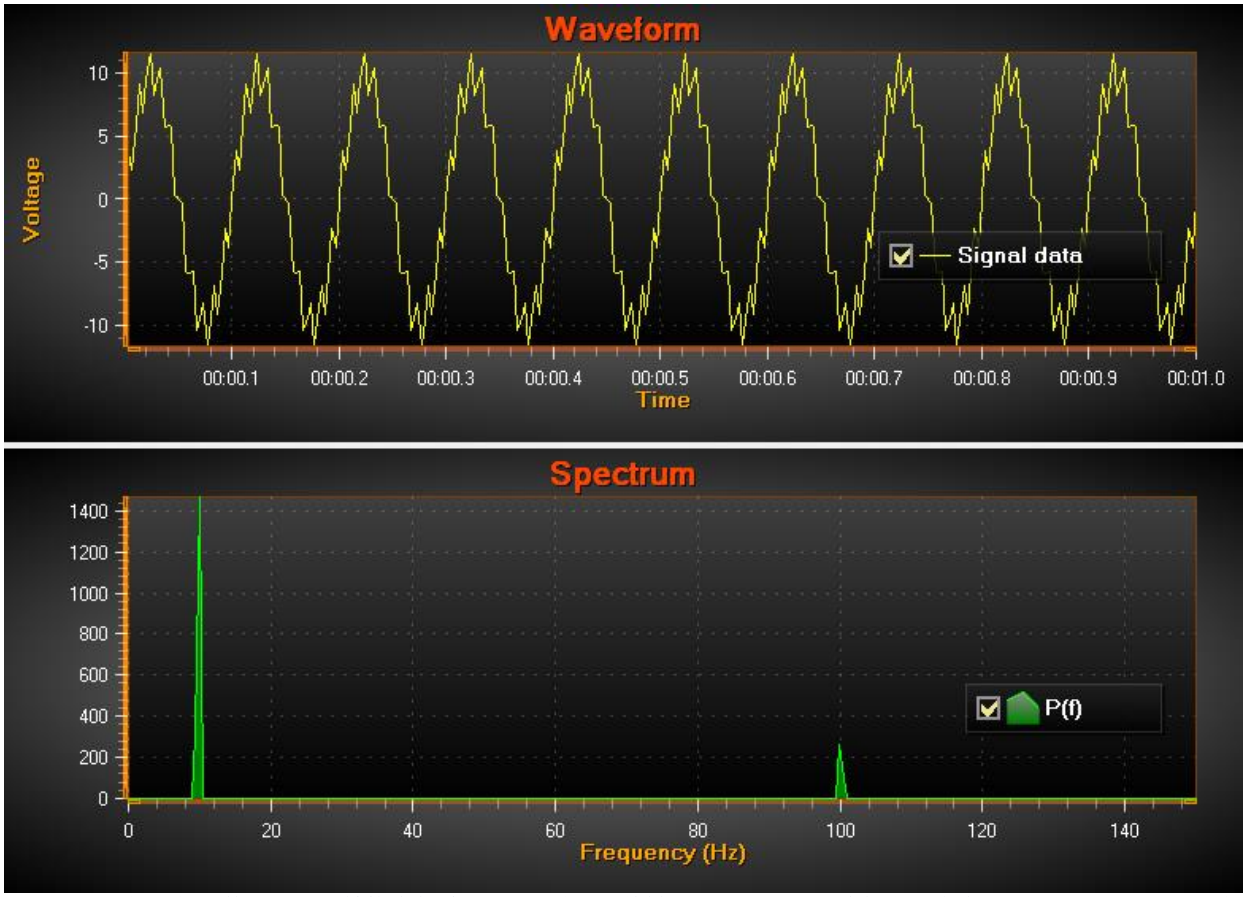


图 233-1. 示例：信源信号数据（上）转换为频域（下）。信号采样频率 = 300 Hz, 因此频率刻度为 300/2 = 150 Hz；强噪声背景下正弦信号的基线是 10 Hz （10 周期/秒）。加入较小的 100 Hz 信号作为噪声。这两个峰值都出现在功率谱中。

以下是可用的公共方法：

- ***CalculateForward*** *（double[] samples, out double[] fftData）*- 利用 FFT 将时域信号数据转换为频域信号。输出 fftData 也包含负值。输入和输出数据数组的长度必须相等。长度是数据的分辨率，从 0 Hz 一直到采样频率/ 2，输出值之间的频率间隔相等。
- ***CalculateForward*** *（float[] samples, out float[] fftData）*– 类似于前一种方法，但针对单精度浮点值。
- ***CalculateBackward*** *（double[] fftData, out double[] samples）*- 将频域数据转换为时域数据。从 FFT 数据中提取信号采样。采样量等于输入 fftData 长度。

- ***CalculateBackward***（*float[] fftData, out float[] samples*）– 类似于前一种方法，但针对单精度浮点值。
- ***PowerSpectrum***（*double[] samples, out double[] fftData*）- 计算信号数据的功率谱。与 ***CalculateForward*** 相同，但具有绝对输出值。
- ***PowerSpectrum***（*float[] samples, out float[] fftData*）–类似于前一种方法，但针对单精度浮点值。
- ***PowerSpectrumOverlapped***（*double[] samples, int fftWindowLength, double overlapPercent, out double[] fftData, out int processedSampleCount*）- 通过改变信源信号采样数据内的计算窗口，按重叠百分比计算功率谱。信号数据必须大于给定的 FFT 窗口长度。输出的 FFT 数据是 fftWindowLength 的长度，不一定与源数据的长度相同。输出数据有绝对值。

24. Signal filters 信号滤波器

信号处理类具有内置的数字信号滤波器。 它们旨在从采集的信号数据中滤除不需要的频率。 有两种类型的过滤器。 有限脉冲响应 (FIR) 滤波器是幅度稳定、恒定相移的滤波器，会导致数据出现恒定滞后。 因子计数（抽头）越高，延迟越长。 无限脉冲响应 (IIR) 滤波器是最小滞后滤波器，但会根据输入频率引入相移，如果设计不当则不稳定。

必须使用 SignalProcessing 命名空间才能使用信号过滤器。 这允许创建 FIRFilter 和 IIRFilter 对象。 要过滤数据，请调用任一对象的 FilterData(rawData, out filteredData) 方法以使用相应的过滤器。

```
using LightningChartLib.Wpf.SignalProcessing;

// Creating a FIR filter and filtersamples with it.
FIRFilter _firFilter = new FIRFilter();
double[] filteredSamples;
_firFilter.FilterData(rawData, out filteredSamples);
```

过滤器类有几种关于它们行为的方法。 两个滤波器的 SetFactor() 和 IIR 滤波器的 SetABFactors() 可用于修改控制因素，例如数据滞后和不稳定的输出限制。 GetDelay() 获取当前数据延迟，而 Reset() 重置内部延迟和过滤缓冲区。

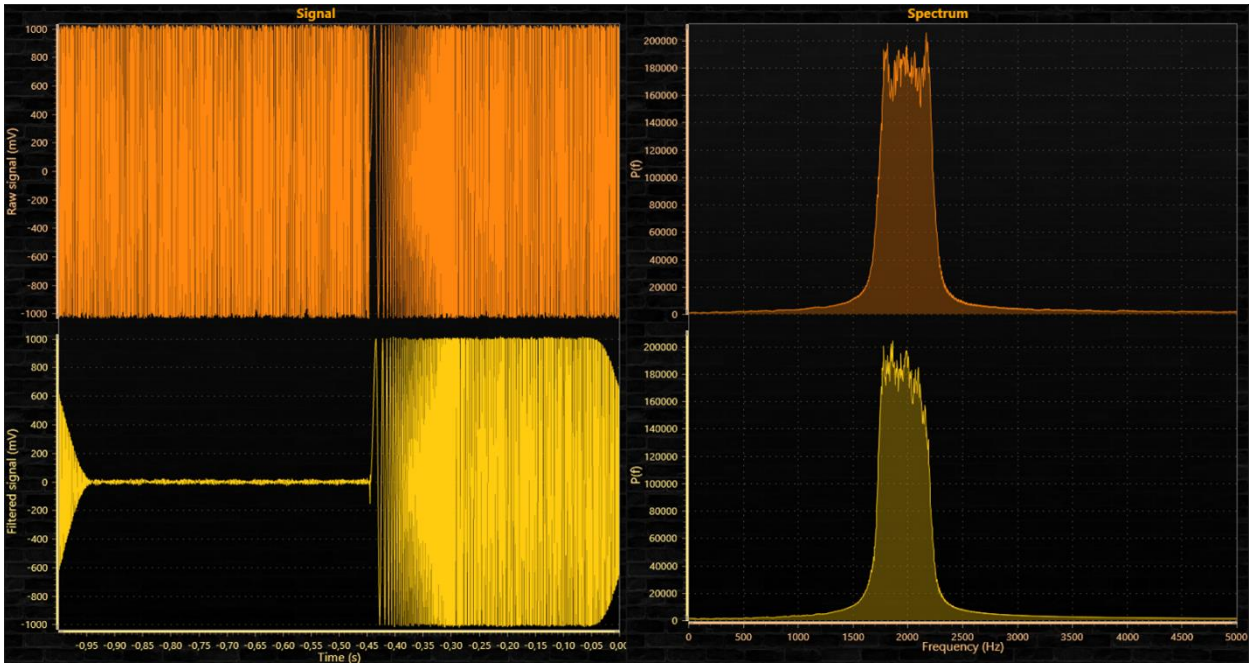


Figure 24-1. 顶部是未经过滤的原始信号，底部是过滤后的信号。

25. Headless mode（无头模式）

无头模式是一种软件能力，可在无法访问图形用户界面（GUI）的设备上工作。当软件不需要外围设备（如显示器、键盘、鼠标）或不需要连接它们时，也会使用“无头”一词。没有外围设备并不会导致初始化或执行过程失败。但是，在这种情况下，软件可以通过例如通过网络或串口等其他通信接口来接收输入并提供输出。

25.1.1 无头渲染

无头配置可以在无头/服务器环境中运行 LightningChart。预期的场景包括在没有用户界面（UI）的软件应用程序中的背景渲染以及根据图表内容生成位图图像。然后图像可以传递到 headful 系统进行进一步的渲染。

基本用法:

```
var chart = new LightningChart（new RenderingSettings（）
{
    HeadlessMode = true
}）;
```

通过设置 ***HeadlessMode*** 标示为 ***true***，可以激活无头模式。该属性可以通过 ***chart.ChartRenderOptions***（用于 WPF）or ***chart.RenderOptions***（用于 WinForms）来获取。LightningChart 自动检测其在 Windows Service 类型应用程序中的使用情况，因此不需要指定模式。

25.1.1.1 其他初始化选项

初始化的 LightningChart 实例缺少 UI 和可视化父类，将不会接收任何渲染请求，比如调整设计大小或何时渲染一个帧。此外，当 WinForms 在创建期间进行引擎初始化时，WPF 图表使用这些信号来初始化渲染引擎。因此，用户必须对图表应用以下操作和配置:

- 使用 ***chart.Width*** 和 ***chart.Height*** 属性来定义尺寸大小。
- 通过调用 ***chart.InitializeRenderingDevice（true）***请求渲染引擎初始化（仅适用 WPF）.
- 订阅 ***chart.AfterRendering*** 事件来实现导出图像的逻辑。

图表仍然对属性更改做出反应。如果需要，可以通过连续的 ***BeginUpdate（）***和 ***EndUpdate（）***调用来查询一个新帧的渲染情况。

25.1.1.2 捕捉图像

已渲染的帧可以通过多种方式导出（参阅第 14 章节）：

- ***OutputStream*** 属性
- ***SaveToStream*** 方法

- ***CopyToClipboard*** 方法
- ***CaptureToByArray*** 方法
- ***SaveToFile*** 方法

一般来说，首选是位图流。另外，ViewXY 图表以无头模式用 ***SaveToStream*** 和 ***SaveToFile*** 方法支持 EMF、WMF、SVG 。

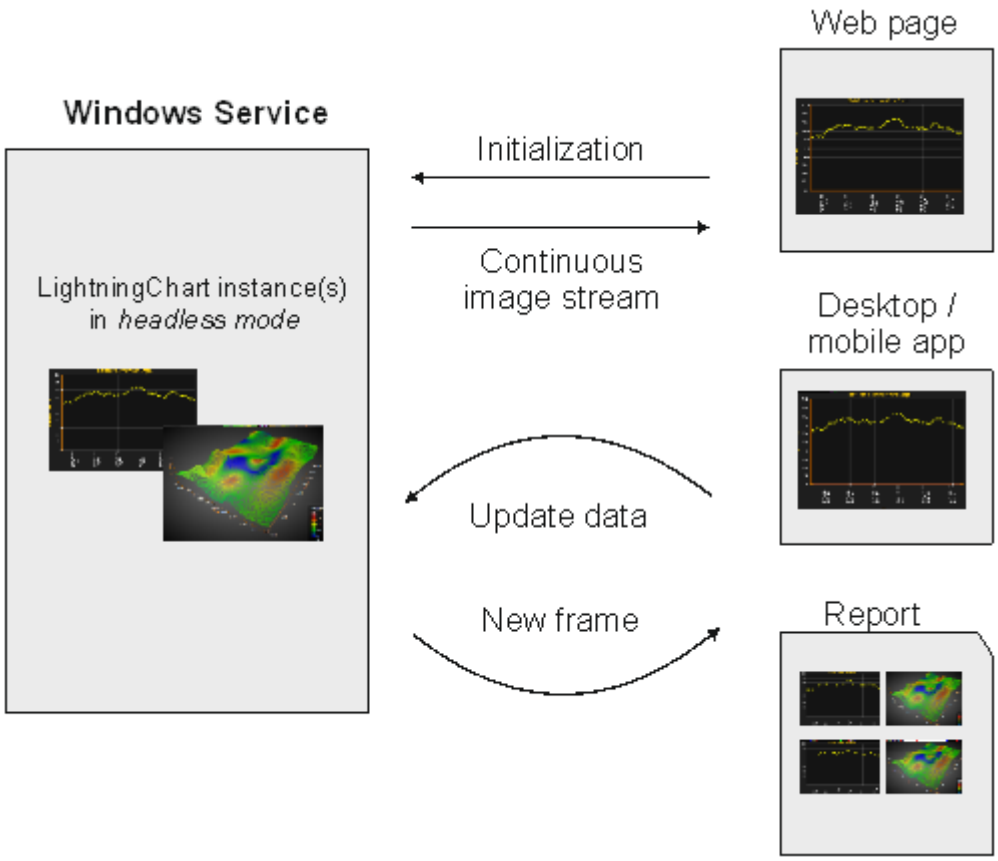


图 254-1. 示例用法图解.

25.1.2 限制和要求

25.1.2.1 阈值

无头配置可以使用 LightningChart 进行后台操作，而不需要将其放在可视化父进程中，也不需要从前台线程（GUI 线程）访问该图表。在创建 LightningChart 实例期间，必须在创建图表的同一线程中更新图表的属性。

- COM 线程模型，又称作“***apartment***”，必须是 STA （单线程套间 Single Thread Apartment），不是 MTA （多线程套间 Multi Thread Apartment）。对于一个普通的 UI 应用程序，STA 是其默认的模型，而对于 Windows Services，MTA 状态则是默认的。
- 所有访问，即更新、创建和处理，都必须通过该线程进行。UI 只能在 GUI 线程中使用。因此，如果需要交互操作，它们应该从图表的线程转移到 GUI 线程。注意！LightningChart 可以在 GUI 线程上运行。
- 线程必须有一个有效的和主动信息队列泵。例如，在线程上运行 ***Application.Run***。

25.1.2.2 图表更新

LightningChart 在渲染时使用一个单独的缓冲区，因此在导出之前的图像后，将处理新图像的导出。同步配置（***ChartUpdateType.Sync***）在接收到更新图表属性的请求后，直接提供图像渲染。同步模式应该为无头模式开启，以促成更快且不间断的性能表现。

25.1.2.3 引擎支持

DirectX 9 和 11 引擎都可在无头模式运行。然而，由于 MS DirectX 的限制，只有 DirectX 11 可以用于 Windows Services 类型的应用程序。

25.1.2.4 许可

默认情况下，Windows Service 在系统用户帐户的安全背景中执行。**不可能安装试用与开发许可。**因此，该服务应用程序**必须**包含一个有效的***部署密钥***，或者使用具有有效许可（试用/开发）的普通用户的认证信息运行。

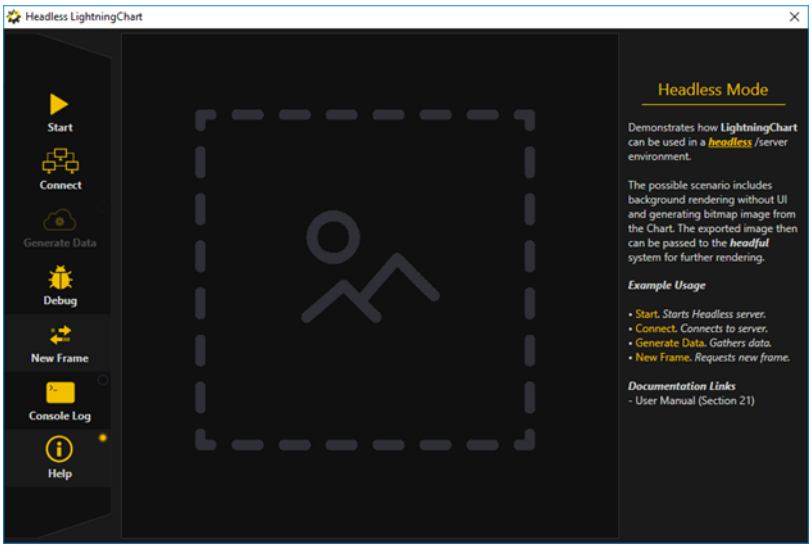
25.1.3 示例解决方案

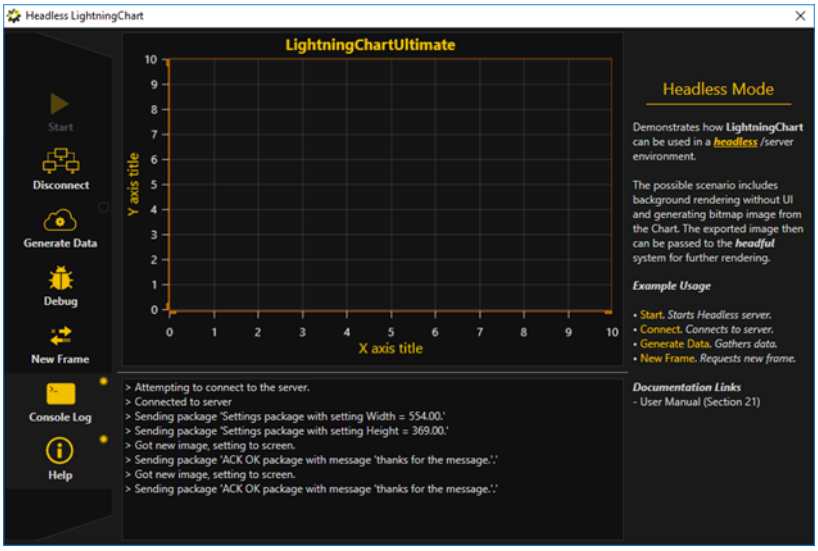
LightningChart SDK 自带有一个 示例 Visual Studio 解决方案（***DemoService.sln***），包含:

- 服务
- 控制台应用程序
- WPF 客户端应用程序

DemoService.sln 可在下面文件夹中找到： ***C:\ProgramData\Arction\LightningChart .NET SDK v.10\DemoService*** .

在启动 WPF 客户端后，它会在窗口中间显示一个帧容器。





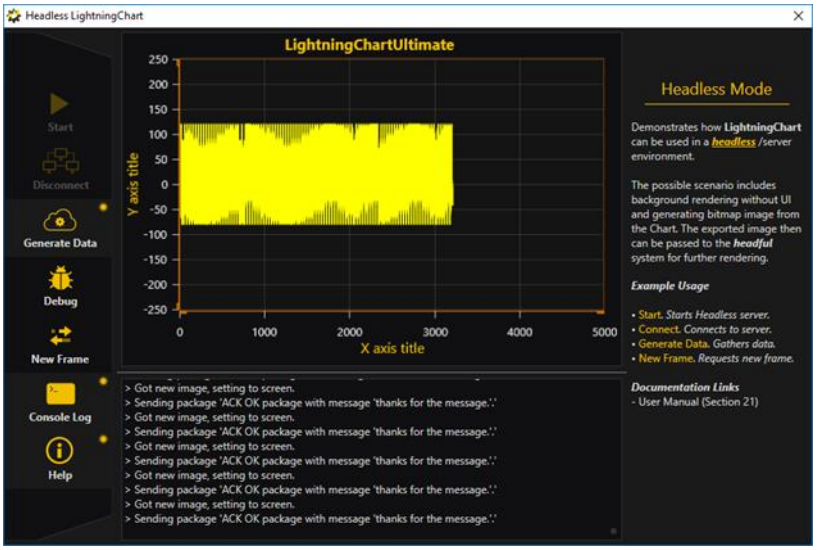
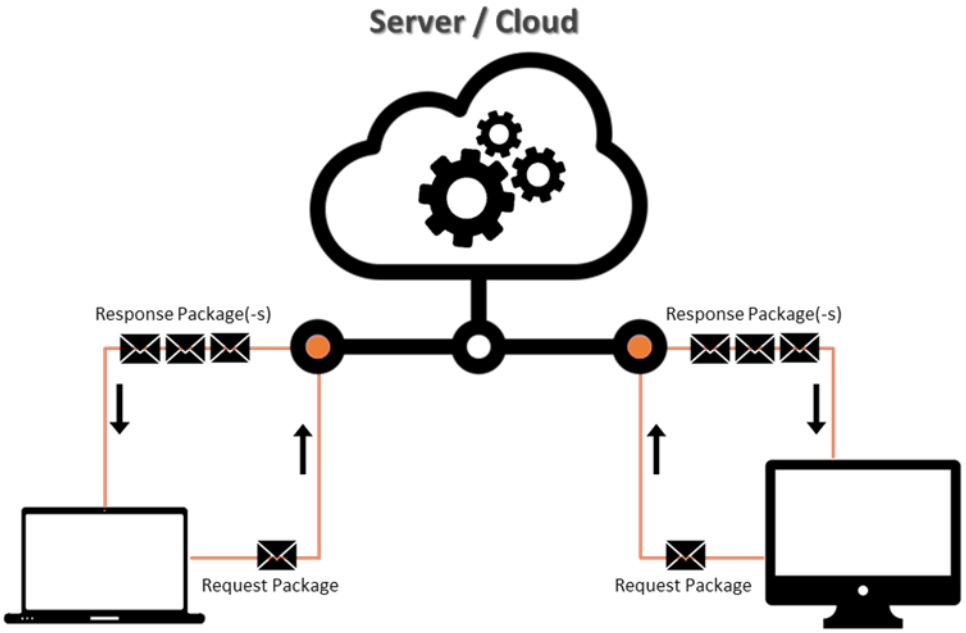


图 254-2. WPF 客户端 app；启动、连接、生成数据后，会显示连续更新的图像流。



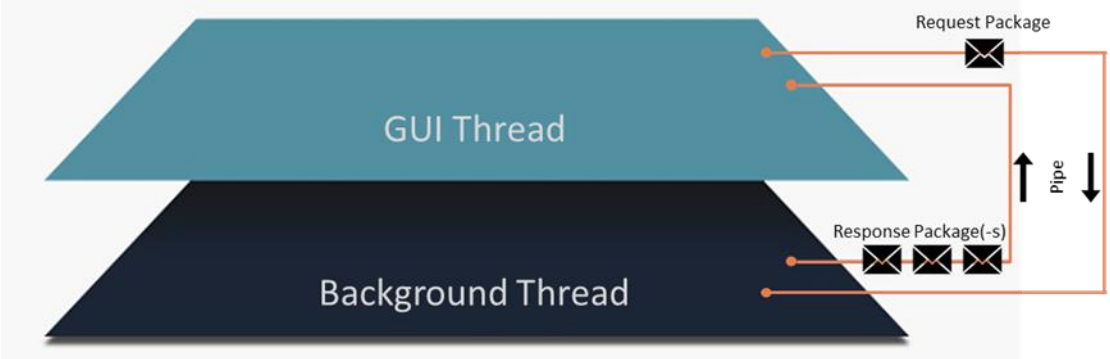


图 254-3. 无头演示服务——消息的客户端内部操作，带有图示说明的命名管道和后台线程。

26. 在 WPF 应用程序中采用 Windows Forms 图表

LightningChart 有 3 个可用的 WPF API。考虑仅在特殊情况下在 WPF 应用程序中使用 WinForms 图表 API。

26.1 在 WPF 中采用 Arction Windows Forms 控件怎么样?

在 WPF 中，可以通过添加 ***Arction.WinForms.Charting.LightningChart.dll 和 Arction.WinForms.SignalProcessing.SignalTools.dll*** 作为项目引用来使用 Windows Forms 组件，并通过代码来创建。 LightningChart 控件以及大多数其他控件都有内置的 UI。使用 WindowsFormsHost 作为其父容器。这些控件也可以在没有 UI 的情况下，采用自己的方法和属性来使用。

26.2 我应该在 WPF 中使用 LightningChart.WinForms 吗？？

在 WPF 应用程序中，推荐使用 WPF 图表程序集，因为它不需要 ***WindowsFormsHost*** 控件，因此不存在 ***WindowsFormsHost*** 控件中常出现的“空域”问题。另一个优点是 WPF 图表可以设置透明的背景，而且可以将一个图表置于另一个上方。

当需要绝对最大性能时，可以考虑使用具有 WinForms 图表控件的 ***WindowsFormsHost*** 控件。***WindowsFormsHost*** + WinForms 图表渲染会稍微块些。

如果用户选择在 WPF 应用程序中使用 WinForms 图表，则必须将其放置在 ***WindowsFormsHost*** 控件中。向 WPF 窗体添加一个 ***WindowsFormsHost*** 控件（在 Visual Studio WPF Toolbox 可以找到）。

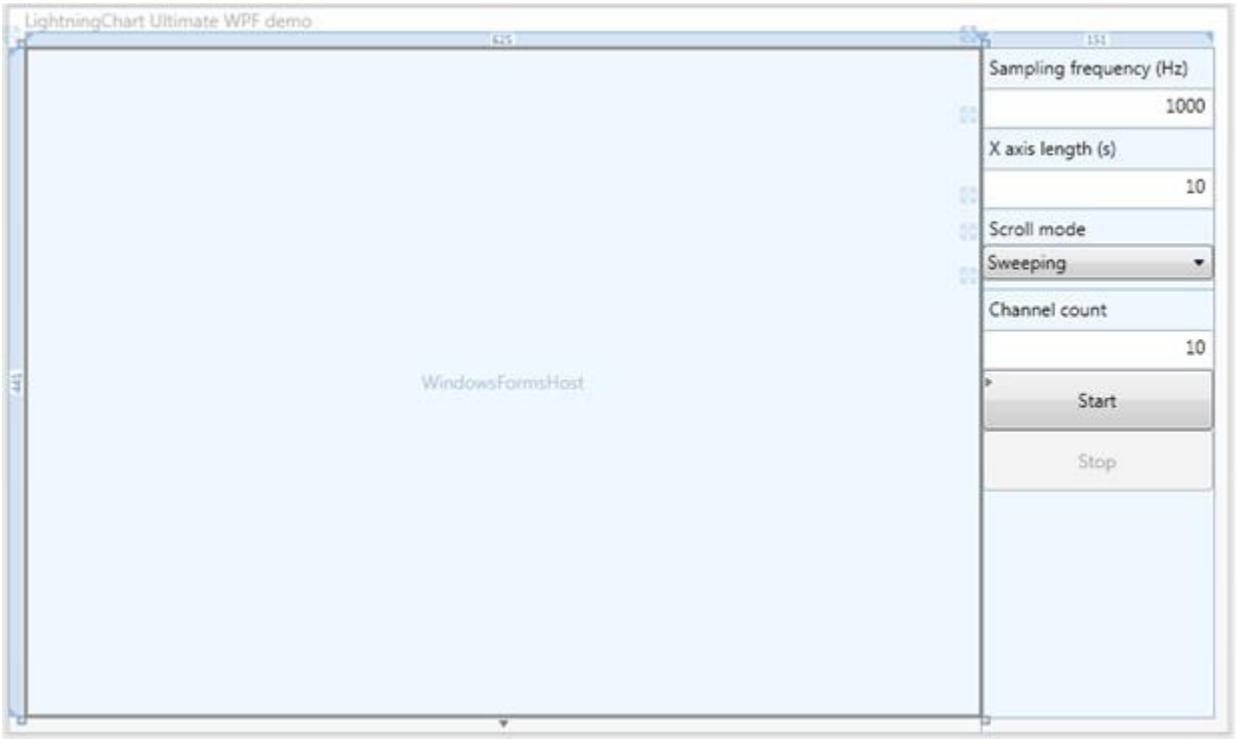


图 265-1. 设计窗中的 WPF 应用程序。当执行应用程序时， WindowsFormsHost 控件保持 the LightningChart 对象在内部。

用代码创建一个 LightningChart 对象，并将其置于 ***WinFormsHost*** 对象中。打开窗体的 xaml.cs 文件，在窗体构造函数中创建图表：

```
public WindowMain ()
{
   InitializeComponent () ;

   CreateChart () ;
}

private LightningChart m_chart = null;

void CreateChart ()
{
        m_chart = new LightningChart () ;

        //将图表对象设置为WindowsFormsHost控件的子对象
        windowsFormsHost1.Child = m_chart;
}
```

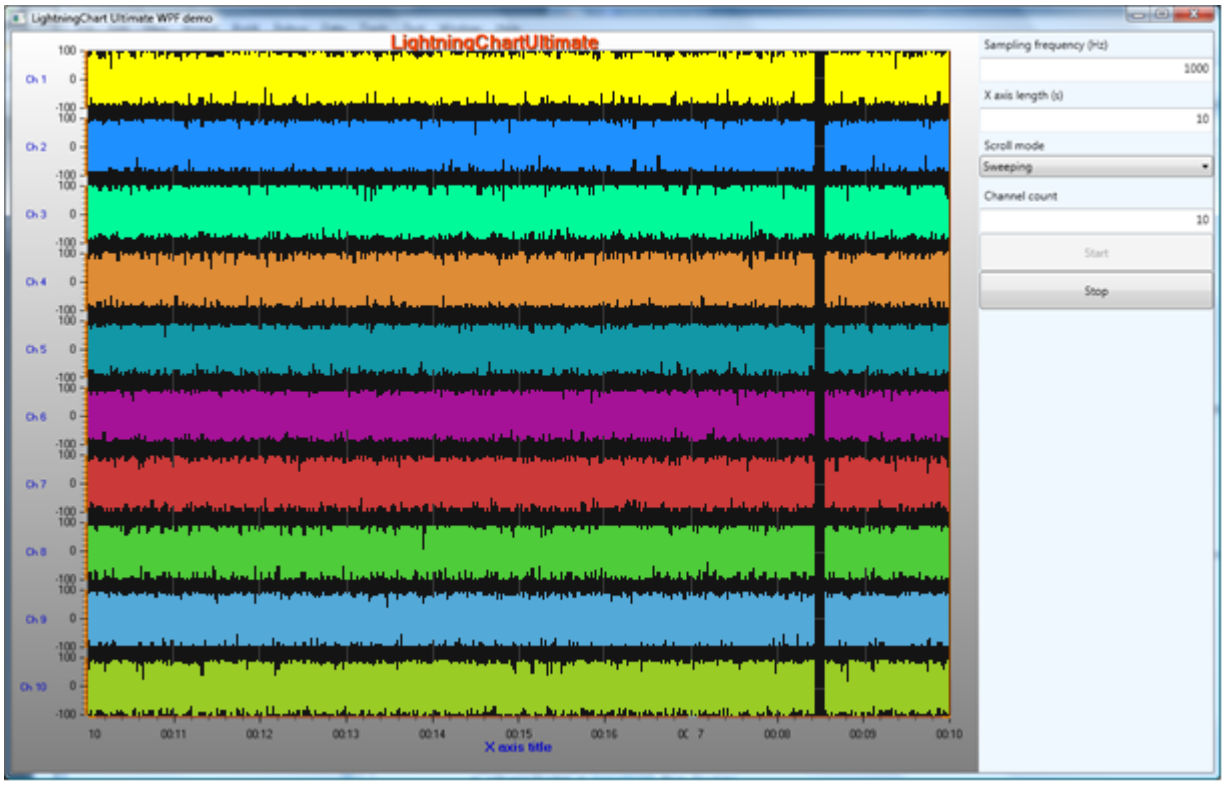


图 265-2. 一个 WPF 应用程序中的 WinForms 图表.

27. 在 C++ 应用程序中使用 LightningChart

LightningChart 是个 .NET 库，可以非常顺畅地与 C# 及 VB.NET 语言使用。而且，也可以在 C++ Win32 应用程序中使用 LightningChart，包括 MFC 应用程序。使用的应用程序必须用 ***Common Language Runtime Support （/clr）***选项进行编译。当要用 C++创建一个 Windows Form Project 项目时，可以参考下面详细的分步教程。

27.1 安装所需的 C++/CLR 程序包

确保你的 Visual Studio 已安装带有 C++/CLR 的 C++ 程序包。例如，运行 Visual Studio （2017）Installer，然后选择更新/修改（update/modify）按钮 。从 Individual 组建中选择 ***C++/CLI support***。从 Workloads 中选择 ***Desktop development with C++***.

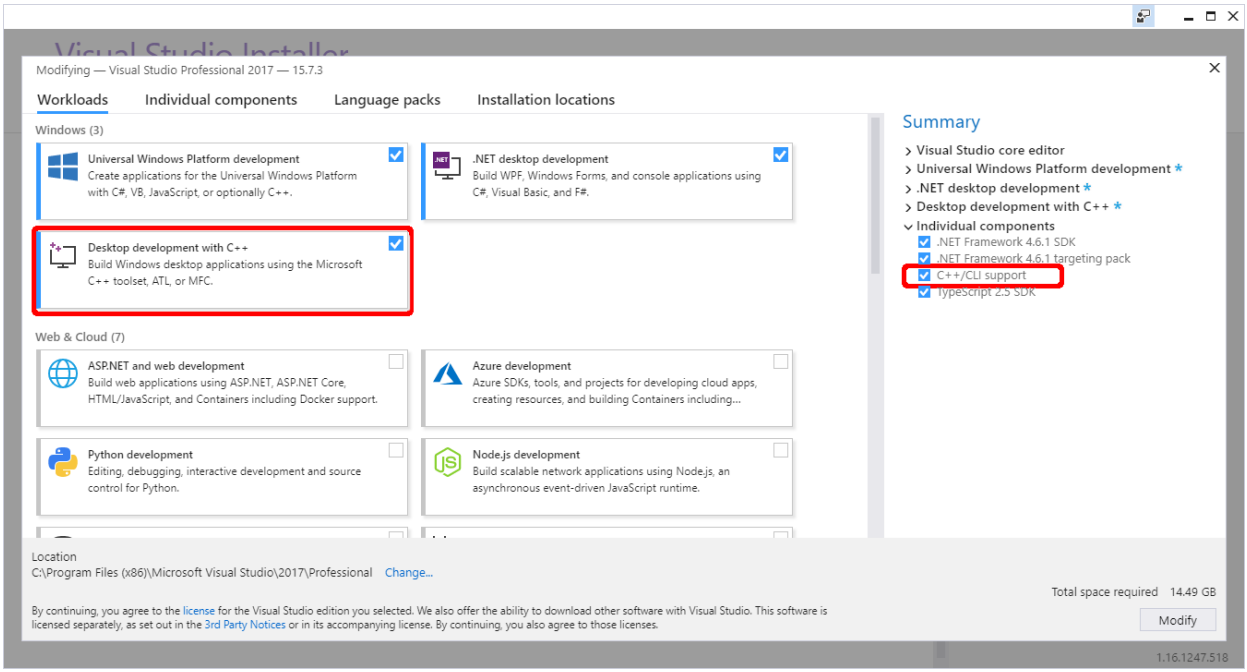


图 276-1. C++ 项目的 Visual Studio Installer 选项.

27.2 设置 Visual Studio 项目

打开 Visual Studio 2017，然后创建一个新项目。如果已安装了上述所有必需的程序包和组件，则在创建新项目时可以选用以下选项（***Templates -> Visual C++ -> CLR -> CLR Empty project***）.

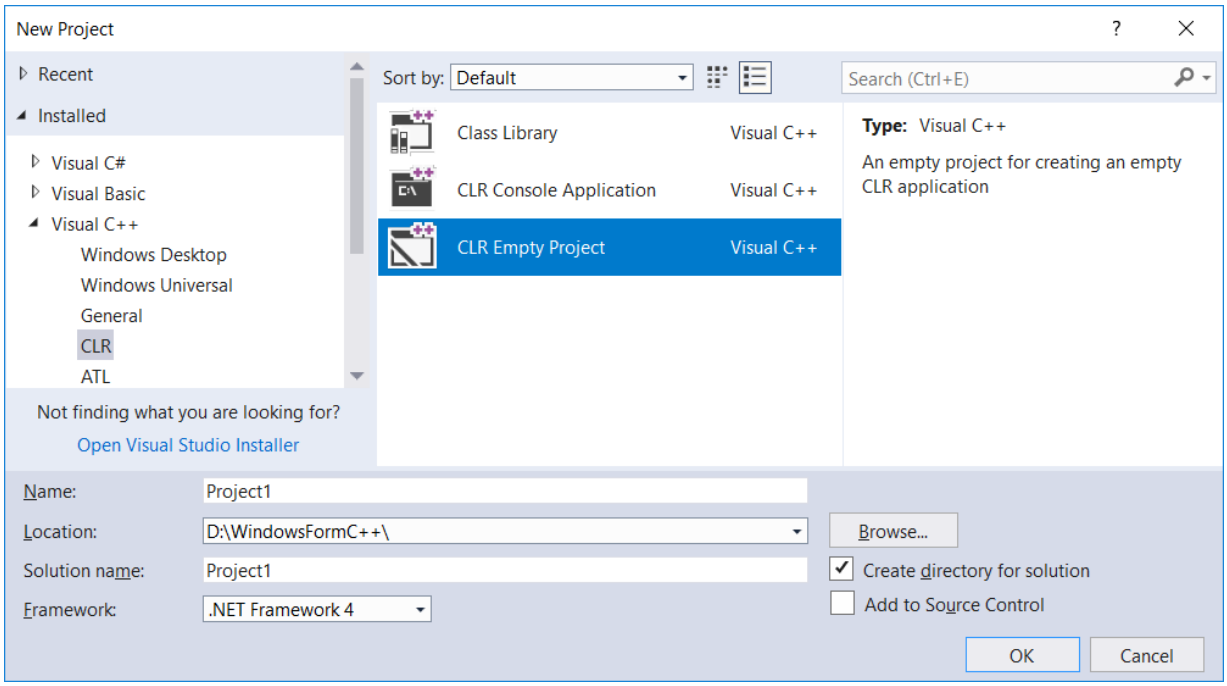


图 27-2. Windows Forms C++ 项目模板

右击所创建的仙姑，然后选择 Properties 选项。 按照图 24-3 所示进行修改 ***Configuration Properties -> Linker -> System -> SubSystem*** 和 ***Linker -> Advanced -> Entry point***

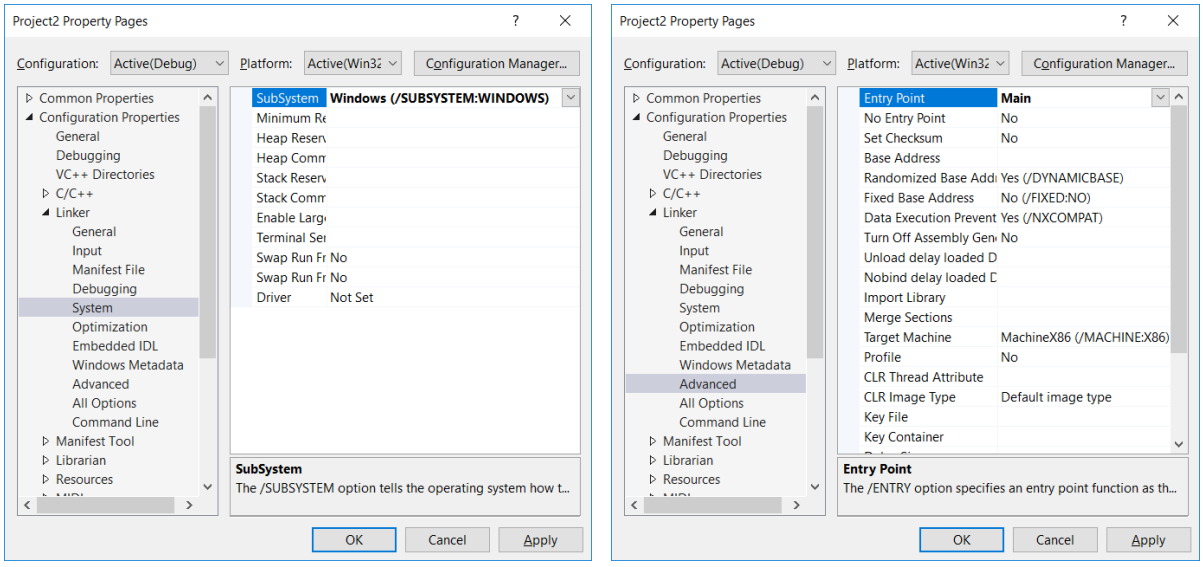


图 27-3. C++项目属性页

向项目添加 *Windows Form* 项：右击项目，然后选择 ***Add -> New Item...*** ，依照图 24-4 所示来选择 Windows Form。这时可能会出现一个出错信息提示：尚未找到完成本操作所需数据（异常 **HRESULT: 0x8000000A**） **The data necessary to complete this operation is not yet available.（Exception from HRESULT: 0x8000000A）**。此信息可以忽视，直接关掉，然后进入到下一步。

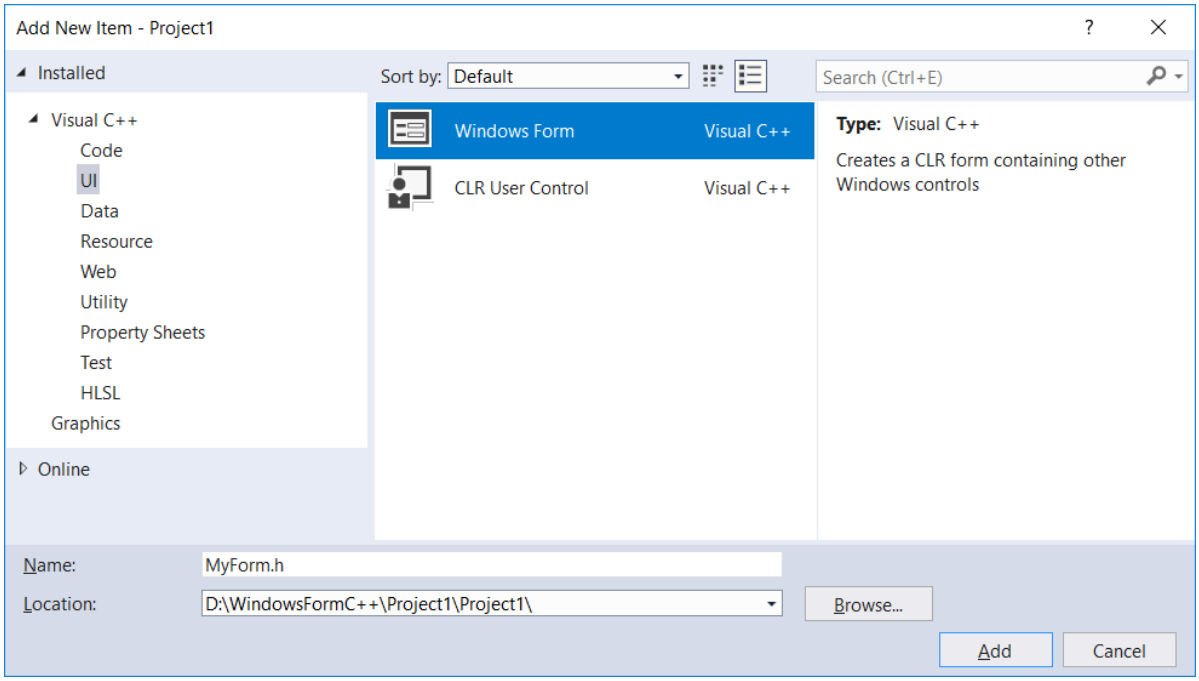


图 276-4.在 C++项目中添加新的 Windows Forms 项

将以下代码添加至所创建的窗体（在本示例 MyForm.cpp），保存然后关闭 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);
    }
```

准备第一次搭建该项目。重新打开该项目，然后选择 ***Build -> Rebuild Solution***。当运行该项目后，应该可以看到一个空的 *Windows Form*.

27.3 在 C++ 项目中创建 LightningChart 应用程序

现在通过编辑 ***MyForm.h*** 文件可以将组建添加至窗体。下面是关于如何创建一个包含 ***PointLineSeries*** 的图表的简单示例。在引用列表中加入 Arction 的 WinForms DLL，并在 MyForm.h 代码文件中添加相关的名称空间。

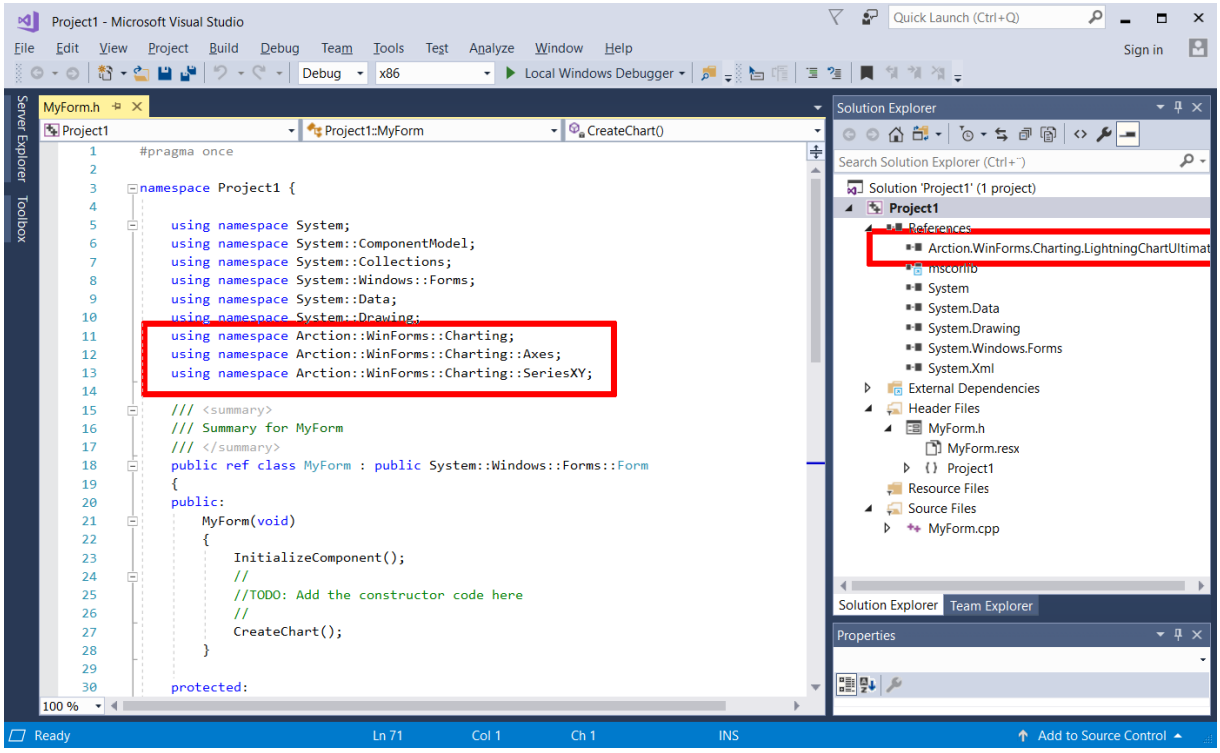


图 27-5. 将 Arction.WinForms.Charting.LightningChart.dll 包含在引用列表中，并在项目中添加相关命名空间。 在版本 11.0+ 中，参考是 LightningChart.WinForms.Charting.NET4。

声明一个‘chart’变量并设置它的属性。下面是一个图表创建方法的例子。

```
protected:
LightningChart ^ _chart;

void CreateChart（）
{
        _chart = gcnew LightningChart（）；

        //为每个属性变更禁用重绘
        _chart->BeginUpdate（）；

        //通过窗口句柄设置父窗口
        _chart->Parent = this;

        //填充窗口区
        _chart->Dock = DockStyle::Fill;

        _chart->ActiveView = ActiveView::ViewXY;

        // 配置X轴
        AxisX^ axisX = _chart->ViewXY->XAxes[0];
        axisX->SetRange（0, 20）；
        axisX->ScrollMode = XAxisScrollMode::None;
        axisX->ValueType = AxisValueType::Number;
```

```
            // 配置Y轴
            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）;

            // 生成随机数据
            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;

            // 允许图表渲染
            _chart->EndUpdate（）;
}
```

编译并执行后生成的应用程序：

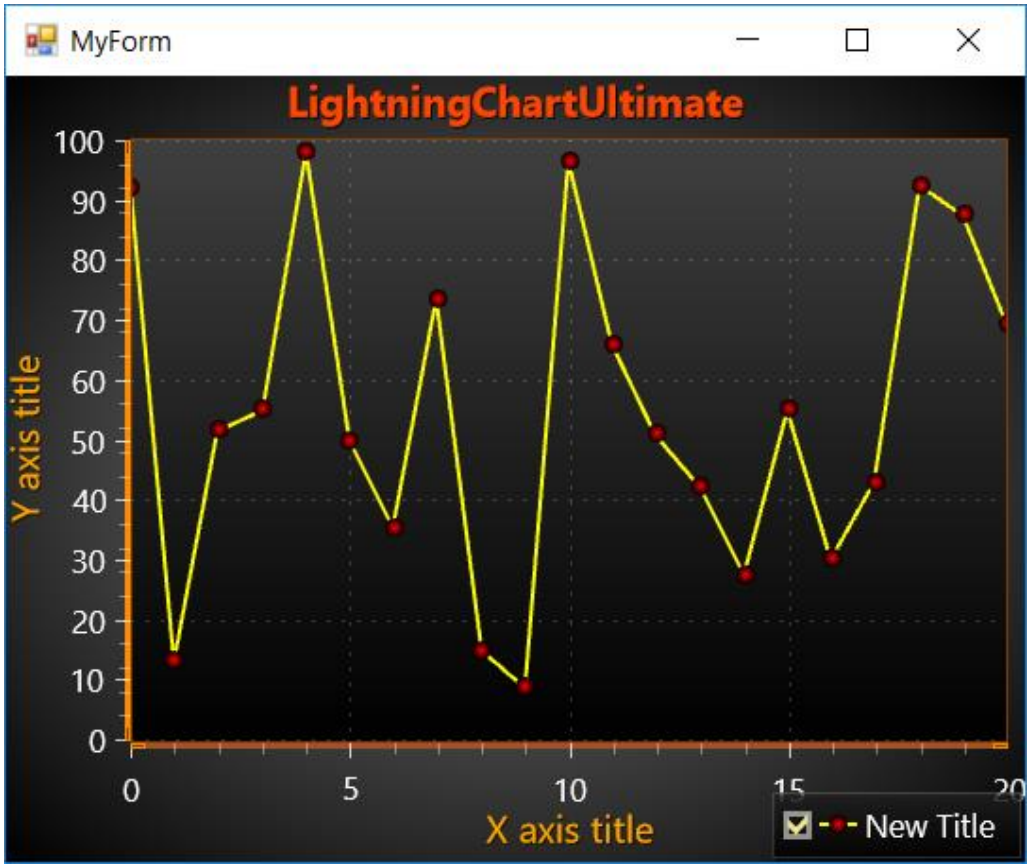


图 276-6.执行的示例应用程序

28. 处理模式

28.1 图表处理

当用代码创建了一个图表，并且不在需要了，应该调用 ***chart.Dispose（）***。它会把图表及其所有对象（例如系列、标记和调色板色阶）从内存中释放掉。

28.2 处理对象

如果是动态创建对象，那么在退出应用程序或用 ***chart.Dispose（）*** 处理整个图表之前需要释放它们所使用的内存，则从对象被添加到的集中删除掉该对象，然后为该对象调用 ***Dispose（）***。

例如，从 ***chart.ViewXY.PointLineSeries*** 集中处理一个系列：

```
// 清除…移除及处理3系列

        _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（）;
```

当不再需要 LightingChart 的对象时，最好将它们释放，以防止内存泄漏。

LightningChart 的集合也有正确处理未使用对象的特定方法。 可以使用 RemoveAndDispose 而不是调用通用 list.Clear() 方法（例如 ViewXY.SampleDataSeries.Clear()）。 例如，要删除所有 SampleDataSeries：

```
        while (_chart.ViewXY.SampleDataSeries.Count > 0)
        {
            _chart.ViewXY.SampleDataSeries.RemoveAndDispose<SampleDataSeries>(0);
        }
```

29. 对象模型说明

29.1 在其他对象之间共享对象

LightningChart 对象模型是基于树型结构的。每个类都有它的父对象和一组子对象。这个树型模型可以将子对象的变更通知给父对象，让父对象进行响应。另外，父节点会再通知它的父节点，以此类推，直到到达根节点 LightningChart 本身，然后可知如何相应地刷新。

图表掌握所有赋予给它的对象的所有权，而且当它不再需要这些对象时，就会释放它们。这包括将一个新对象替换一个旧对象，并将其父对象丢弃的情况。用户必须意识到这一点，否则可能最终使用已处理的对象。

如果一个对象在其他的.NET 组件和 LightningChart 之间共享，并且 LightningChart 处理该对象，那么.NET 组件就会剩下一个已处理的对象。LightningChart 无法检测到 LightningChart 与其他组件之间的父组件共享情况。

不允许在同一图表或其他图表实例中的其他对象之间共享对象。

错误用法示例 1:

```
AnnotationXY annotation1 = new AnnotationXY（）;
chart.ViewXY.Annotations.Add（annotation1）;

AnnotationXY annotation2 = new AnnotationXY（）;
annotation2.Fill = annotation1.Fill;
chart.ViewXY.Annotations.Add（annotation2）;
```

问题: 同一个 ***Fill*** 对象不能在多个对象之间共享。
纠正方法：如果他们是 ValueType （例如 Integer, Double, Color），则只复制属性

错误用法示例 2：

```
SeriesEventMarker marker = new SeriesEventMarker（）;

chart.ViewXY.PointLineSeries[0].SeriesEventMarkers.Add（marker）;
chart.ViewXY.PointLineSeries[1].SeriesEventMarkers.Add（marker）;
```

问题： 同一个对象不应该被添加到多个集合的某个集中。
纠正方法：创建几个标记，每个系列一个.
记得订阅 ChartMessage 事件处理程序。在大多数情况下，它会报告无效对象共享情况中的错误（参阅第 16 章节）。

30. LightningChart 程序集的发布/分发

30.1 引用的程序集

使用可执行文件夹、可执行文件夹下文件夹、全局程序集缓存或 NET 程序集解析系统可以找到它们的另一个文件夹来转发 Arction .dll 文件。

LightningChart 还支持 ***ClickOnce*** 发布。

WinForms:

- LightningChart.WinForms.Charting.NET4.dll **(in .NET Framework projects)**
- LightningChart.WinForms.Charting.NET6.dll **(in .NET6+ projects)**
- LightningChart.DirectX.dll
- LightningChart.DirectXInitNET.dll
- LightningChart.DirectXFilesNET.dll

如果使用 SignalTools

- LightningChart.WinForms.SignalTools.NET4.dll **(in .NET Framework projects)**
- LightningChart.WinForms.SignalTools.NET6.dll **(in .NET6+ projects)**
- LightningChart.MathCore.dll

WPF:

- LightningChart.WPF.Charting.NET4.dll **(for Non-bindable WPF chart, .NET Framework)**
- LightningChart.WPF.Charting.NET6.dll **(for Non-bindable WPF chart, .NET6+)**
- LightningChart.WPF.ChartingMVVM.NET4.dll **(for Bindable WPF chart, .NET Framework)**
- LightningChart.WPF.ChartingMVVM.NET6.dll **(for Bindable WPF chart, .NET6+)**
- LightningChart.DirectX.dll
- LightningChart.DirectXInitNET.dll
- LightningChart.DirectXFilesNET.dll

如果使用 SignalTools

- LightningChart.WPF.SignalTools.NET4.dll **(in .NET Framework projects)**
- LightningChart.WPF.SignalTools.NET6.dll **(in .NET6+ projects)**
- LightningChart.MathCore.dll

UWP:

- LightningChart.UWP.ChartingMVVM.dll
- LightningChart.UWP.Attributes.dll
- SharpDX.D3DCompiler.dll

- SharpDX.Direct2D1.dll
- SharpDX.Direct3D11.dll
- SharpDX.dll
- SharpDX.DXGI.dll
- SharpDX.Mathematics.dll

如果使用 SignalTools

- LightningChart.UWP.SignalTools.dll

30.2 许可密钥

记住，对所有组件使用静态 ***SetDeploymentKey*** 方法。否则，图表会进入试用模式，只可使用 30 天，并会不断有催付费提醒。*有关部署密钥和详细的许可证密钥管理信息，请参阅第 4 章。*

30.3 应用程序代码模糊

必须**强制**模糊应用程序代码，这样 LightningChart 许可证密钥才可以对.NET 反汇编工具不可见。泄漏许可证密钥可能导致许可证终止、法律诉讼和损害索赔。

30.4 LightningChart 代码模糊

LightningChart 源代码订购者可以访问 LightningChart 库的源代码。必须**强制**模糊根据 LightningChart 源代码构建的程序集，以防止 Arction 的知识产权和代码泄漏。分发未模糊的 LightningChart 库违反了最终用户许可协议（EULA），可能导致许可证终止、法律诉讼和损害索赔。

30.5 Arction 程序集的 XML 文件

禁止发布以下 XML 文件。

- LightningChart.WinForms.Charting.NET4.xml
- LightningChart.WinForms.Charting.NET6.xml
- LightningChart.WPF.Charting.NET4.xml
- LightningChart.WPF.Charting.NET6.xml
- LightningChart.WPF.ChartingMVVM.NET4.xml
- LightningChart.WPF.ChartingMVVM.NET6.xml
- LightningChart.WPF.SignalTools.NET4.xml
- LightningChart.WPF.SignalTools.NET6.xml
- LightningChart.WinForms.SignalTools.NET4.xml
- LightningChart.WinForms.SignalTools.NET6.xml

Arction 提供的文件仅用于帮助开发应用程序。它们主要用于显示代码参数和属性建议。根据源代码重新构建 LightningChart 程序集时，请确保没有发布上面提到的 XML 文件。**这些 XML 文件严格禁止发布**，因为它们会向.NET 反汇编程序和逆向工程应用程序泄露太多的信息。

31. 疑难解答

31.1 旧版本更新

LightningChart 组件 API 可能已经变更了旧版本，之后的项目可能不会自动加载或使用新版本。这些说明介绍了如何对新版本程序集进行设置以引用给一个项目，以及如何修复 Visual Studio 窗体编辑器中无法反序列化的属性。

为了更新图表，必须删除对旧版本的引用并添加对新版本的引用。在某些情况下，可能需要修复*.Designer.cs 和*.resx 文件，因为它们可能包含不兼容二进制的属性。

从项目 References 中移除旧引用

1. 进入 Solution Explorer.
2. 打开 References 文件夹.
3. 选中 Arction 程序及，点选 ***Delete*** 按钮或右击下再选择 ***Remove*** 来删除。

向新版本添加一个引用

1. 进入 Solution Explorer.
2. 打开 References 文件夹.
3. 向新图表添加引用。右击 References 文件夹；选择 ***Add Reference...***，然后选择新 Arction DLL 文件。

由于 API 可能已经更改，所以更改的属性的源代码可能必需要修复。

如果某图表完全不兼容（例如 Visual Studio 无法将 UI 加载到窗体编辑器上），必须从*.Designer.cs 和*.resx 文件中删除 LightningChart 属性设置器。

从*.Designer.cs 文件移除性设置器

1. 用文本编辑器打开*.Designer.cs 文件 （如果可能的话，使用除 Visual Studio 之外的其他编辑器）.
2. 查找然后删除含有 LightningChart 设置器的行。例如：
   this.m_chart.Background =
   ((Arction.LightningChart.Fill)(resources.GetObject("m_chart.Background")));

不需要删除例如位置（Location）和大小（Size）等继承属性。删除通过例如上面"NN = ((...)(resources.GetObject("..."))));"方法而从资源中读取的属性。

从*.resx 文件移除序列化项

1. 用文本编辑器打开*.resx 文件.
2. 找到含有 Arction 对象的 xml 标签（它们由图表成员名称标识。例如"m_chart" 或"LightningChart1"）.
3. 删除包含<data>标签到 xml 对象末尾的行（</data> tag）.
   例如，如果将图表背景序列化为以下 xml 对象，则应该从*.resx 中删除所有以下行：

```
<data name="m_chart.Background" mimetype="application/x-
microsoft.net.object.binary.base64">
<value>
            AAEAAAD/////AQAAAAAAAAAMAgAAAGRBcmN0aW9uLkxpZ2h0bmluZ0NoYXJ0VWx0aW1hd
            GUsIFZlcnNp
            b249NC42LjEuMjAwMSwgQ3VsdHVyZT1uZXV0cmFsLCBQdWJsaWNLZXlUb2tlbj03MmY1N
            WZiZDY5MDFm
            ... lots of encoded stuff ...
             YXlvdXQBAAAAB3ZhbHVlX18ACAIAAAAAAAAACw==
</value>
</data>
```

请注意，有些对象可能非常大，例如标题行数可能约为 200 行，而视图则通常要更加大得多（View3D 有大约 2000 行）。

如果有几个图表的情况，则需要删除它们的所有序列化属性。编辑器搜索是一种十分方便的查找图表对象工具。

从*.resx 文件删除对象，以及从*.Designer.cs 文件移除相关设置器后，应该可以在 Visual Studio 窗体编辑器中成功打开项目。

31.2 网站支持

浏览 www.lightningchart.com/support 可了解更多支持服务信息。

登录 http://www.lightningchart.com/forum 可加入论坛讨论组了解更多信息。

31.3 在虚拟机平台上运行

LightningChart 自带 DirectX10/11 WARP 渲染，用于不允许访问图形硬件的系统。由于 WARP 渲染是在 CPU 中进行的，所以与硬件渲染相比，性能可能会降低。这需要一个支持 DirectX11 的操作系统。

对于不支持 DirectX11 的系统，LightningChart 会返回到 DirectX9 Reference Rasterizer 模式。 性能非常差，只发挥很小一部分的 WARP 性能。对于自动回退到 WARP 和 DirectX9 的，要继续把 RenderDevice 设置为 ***Auto***、***AutoPreferD9*** 或 ***AutoPreferD11***（参阅第 5.121 章节）。

32. 开发组

32.1 Intel 数学核心函数库（Intel Math Kernel library）

LightningChart® .NET SDK 在某些部分使用 Intel 数学核心函数库（Intel Math Kernel library），例如 Fast Fourier Transform 方法。Arction 程序集包含一些从这个库构建的本地 DLL 文件。 Arction Ltd. 获得有使用 Intel 数学核心函数库的许可。

32.2 开源项目

特此鸣谢以下开源项目和资料供应商:

DirectX library for .NET
LightningChart 使用具有 Arction 扩展名的衍生自 SharpDX 的 DirectX .NET DLL 文件,
http://www.sharpdx.org/

Map sources
LightningChart® .NET 地图已从如下地图供应商导入：

World, North America, Europe:	Natural Earth, http://www.naturalearthdata.com/
Australia:	Australian Bureau of Statistics, http://www.abs.gov.au/
Roads of USA:	National Atlas of the United States, http://www.nationalatlas.gov

Scalable Vector Graphics 导出
LightningChart SVG 导出采用部分由 RiskCare Ltd 开发的 SvgNet 项目代码。

多项式回归（Polynomial regression）
多项式回归计算代码部分基于 Math.Net 库开发。 http://www.mathdotnet.com/

修改后的源代码部分可以从 Arction Support 服务免费获取（support@lightningchart.com）.

有关开源项目的版权声明，请参阅 LightningChart SDK 安装文件夹中的 **LightningChart .NET Readme.txt**。