JavaScript Charts Performance Comparison / Line Charts

Test date May 5th, 2021 | Latest update: December 1st, 2021

Foreword

This test is a performance comparison between 12 different charting providers. The charting libraries included in this test are from major manufacturers who classify their charts as high-performance or the fastest. Additionally, this performance comparison considers some open-source and performance-oriented libraries.

We have excluded other charts that are either end-of-life, not supported anymore, or simply do not work. We are confident we have selected all the fastest charts for this comparison, and if we did not, inform us and we will incorporate them in this test.

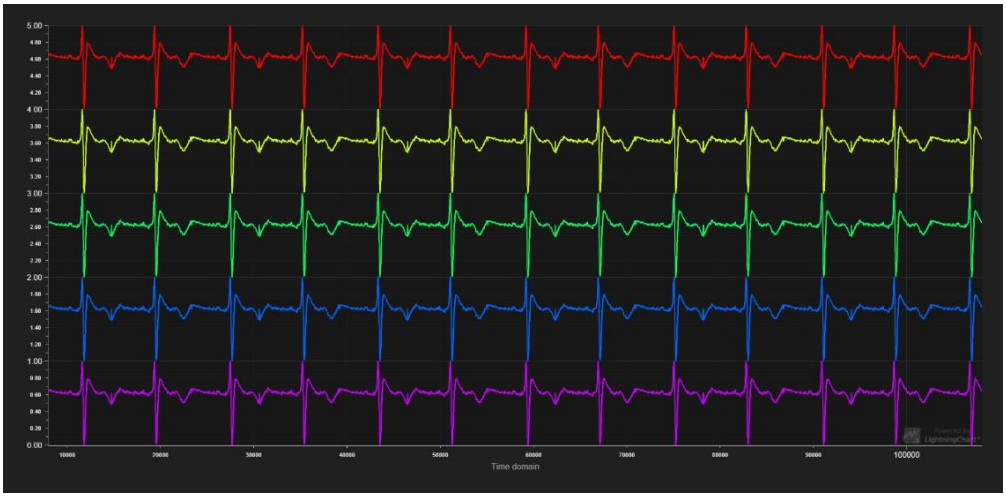
The bias in this test is in progressive line charts, needed typically in medical applications (ECG / EEG / ExG), seismography, telemetry, industrial automation, vibration research and audio engineering, and real-time trading data applications.

We have identified three different application types of line charts which are Static line charts, Refreshing line charts, and Appending surface charts. Each type of line chart has its own performance metrics that set the success/failure between libraries. The libraries tested (in no particular order) were:

- LightningChart JS v.3.3.0
- Highcharts 9.1.0
- SciChart JS v.2.0.2115
- Anychart 8.9.0
- amCharts 4
- ECharts 5
- DvxCharts 5.0.0.0
- Dygraphs 2.1.0
- Canvas.js 3.2.16
- µPlot 1.6.17
- Plotly.js 1.58.4
- ZingChart 2.9.3

Test procedure

The test project is published as open-source project in GitHub.



The goal was to create a scrolling line chart. Data is appended to the tail of the line series, and X axis then scrolled to show the new data, and make the oldest data fall out.

The tests were made according to examples and tutorials and contacting manufacturers' support teams whenever available. We strongly believe all of them have been made in optimal way for all of them, and if you notice any problem or wrong configuring of them, please contact us for a fix.

The appearances of the chart were set to simplistic, and line series stroke widths were set to 2 pixelswide, which is a typical value for line charts.

Test hardware setups

The test was carried out with the following setups.

1. Mid-level Desktop PC
    - CPU: Intel Core i7-7700K
    - GPU: AMD Radeon R9 380
    - RAM: 16 GB

Tested chart libraries (in no particular order)

- LightningChart® JS 3.3.0
- AmCharts v.4
- ZingChart 2.9.3
- HighCharts v.9.1.0
- DvxCharts v.5.0.0.0
- Dygraphs v.2.1.0
- µPlot v.1.6.17
- AnyChart v. 8.9.0
- ECharts v.5
- Plotly.js v.1.58.4
- SciChart JS v.2.0.2115
- Canvas.js v.3.2.16

What did we measure? - Static Line Charts

1. **Load-up Speed**

Measured in milliseconds, this is a metric that tells how many milliseconds, once initiated the rendering procedure, does the chart take to be fully visible to the user.

2. **Frames Per Second (FPS)**

How many times the visualization is updated per second. For good real-time performance should be at least 40 or more.

3. **CPU usage %**

This is a % value between 0 and 100, which indicates how much processing power the chart is using. Lower values are better.

Results

The results were output to Google Chrome browser Console, and constructed into CSV data, and imported to Excel.

The green color represents good value, yellow mediocre, and red... it indicates either slowness, or bright red a fail.

The FPS rates are visualized with bars inside the cells, too.

Static line chart performance comparison breakdown

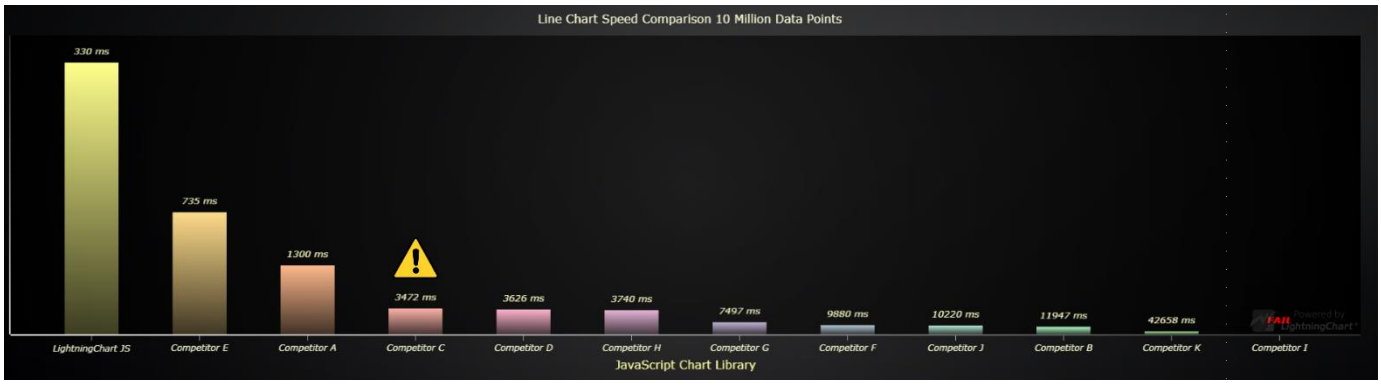
In static data visualization, the most important **measurable** performance attribute is how fast the chart is displayed to the user. Another performance metric, which is not covered by this study is how well the user can interact with the produced chart.

We have selected a single test from the set of static performance tests. This test was the same for each library and it highlights the performance differences most effectively. Here are the results of 10 million data points static line chart test.

JavaScript Chart Library	Loading speed *
LightningChart JS	**330 ms**
Competitor E	735 ms
Hardware accelerated competitor A	1300 ms
Hardware accelerated competitor D	3626 ms
Competitor H	3740 ms
Competitor G	7497 ms
Competitor F	9880 ms
Competitor J	10220 ms
Competitor B	11947 ms
Competitor K	42658 ms
Competitor I	Fail
Competitor C	3472 ms **

* Average of measurements with Google Chrome and Mozilla Firefox browsers.

** Chart library uses down-sampling, the produced data visualization is clearly incorrect.



From the bar chart above, we can see that LightningChart JS is the fastest JavaScript chart in visualizing 10 million data points, being ready **7.5x faster** than the average hardware accelerated chart and **65.7x faster** than the average non hardware accelerated chart.

This is a good place to explain what does the **"loading speed"** measurement include. You might run into various claims of JavaScript loading speed in the internet, but we believe that there is only one correct way to measure this.

Loading speed is the time (seconds) which user has to wait for their chart to be visible on the web page.
Some inconsistencies to this statement which you might have to look out for:

- Setting up rendering frameworks and licenses, or any other steps which users have to do are included in loading time.
    - For example, some manufacturers have omitted the initialization time of graphics engines from loading time, which doesn't make any sense from the perspective of the user and provides false results.
- Loading speed includes any chart processing time between initiating the chart creation and displaying it.
    - We have also identified loading speed claims which disregarded the processing time of chart method calls, once again producing completely irrelevant performance measurements.
- In addition to this, loading speed **also includes any extra time that is required before the chart is visible**.
    - Most JavaScript chart libraries have some internal events which can be used to track when the chart is done with processing data - this however, by no chance means that the data is visible to the user.
- Some competitors also use "subsequent frame render time" as the claimed loading speed, which once again makes no sense from the users' point of view.

Refreshing line chart performance comparison breakdown

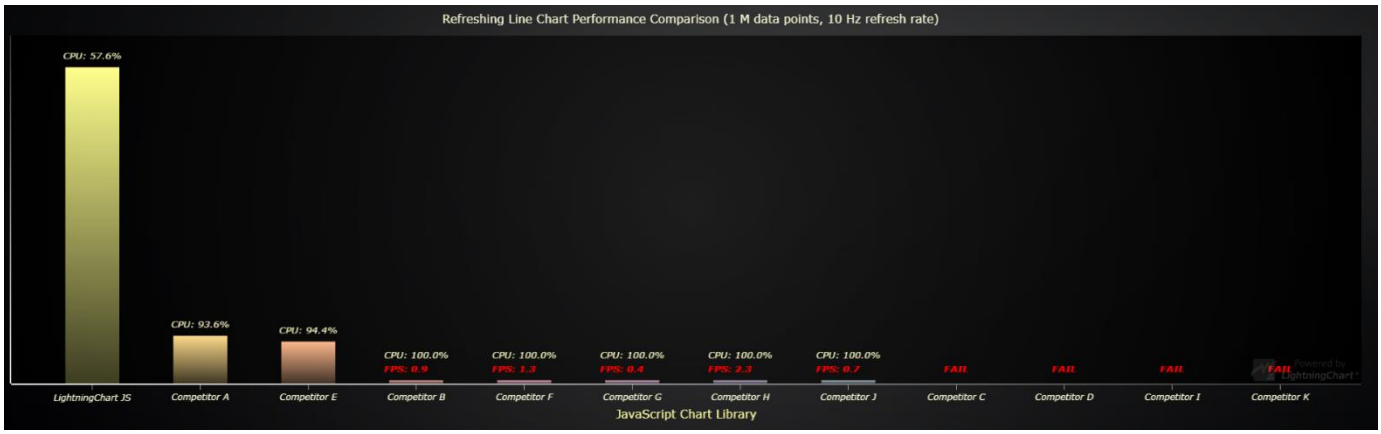
In refreshing chart applications, performance is measured as **refresh rate** (how fast data set can be refreshed, faster is better, unit is expressed as frequency Hz which means how many refreshes per every second) and **CPU usage** (% of processing power used, 0-100).

In web data visualization, the CPU usage measurement is perhaps the most important performance metric which can be measured. This is because almost exclusively all processing on a web page is run in a single process and multiple CPU cores can't be easily utilized. In practice, this means if your web page has a single component which uses CPU extensively it will **ruin the performance of the entire web page**.

If your web page has a chart component which uses 100% of CPU, you can say goodbye to your good user experience.
We have selected a single test from the set of refreshing performance tests. This test was the same for each library and it highlights the performance differences most effectively.

Here are the results of refreshing (refresh rate = 10 Hz) line chart test with 1 million data points

JavaScript Chart Library	Actual refresh rate /s	CPU Usage (%)
LightningChart JS	10.0	57.6 %
Hardware accelerated competitor A	10.0	93.6 %
Competitor E	10.0	94.4 %
Competitor H	2.3	100.0 %
Competitor F	1.3	100.0 %
Competitor B	0.9	100.0 %
Competitor J	0.7	100.0 %
Competitor G	0.4	100.0 %
Competitor C	Fail	Fail
Hardware accelerated competitor D	Fail	Fail
Competitor I	Fail	Fail
Competitor K	Fail	Fail



To help understand viewers to understand the effects of bad refresh rate and CPU usage measurements we have created a YouTube video showcasing the charts mentioned here undertaking the refreshing line chart performance test (not necessarily with same parameters as the test case highlighted above!). In this video you can visible see how a low FPS looks on a web page, and respectively how a good FPS looks.

On average, LightningChart JS could process **14.2x** more data than non-hardware accelerated charts and **9.1x** more data than other hardware accelerated charts.

JavaScript Chart Library	Max data process speed	Data points	Achieved refresh rate *
LightningChart JS	**34 M/s**	8 000 000	4.3 Hz
Competitor E	16 M/s	4 000 000	4.0 Hz
Hardware accelerated competitor A	7.4 M/s	2 000 000	3.7 Hz
Competitor H	2.3 M/s	1 000 000	2.3 Hz
Competitor F	1.3 M/s	1 000 000	1.3 Hz
Competitor B	850 k/s	1 000 000	0.9 Hz
Competitor J	700 k/s	1 000 000	0.7 Hz
Competitor G	400 k/s	1 000 000	0.4 Hz
Competitor C	39 k/s	10 000	4.5 Hz
Hardware accelerated competitor D	39 k/s	10 000	3.9 Hz
Competitor I	14 k/s	10 000	1.4 Hz
Competitor K	None **		

* Average result of Google Chrome and Mozilla Firefox.

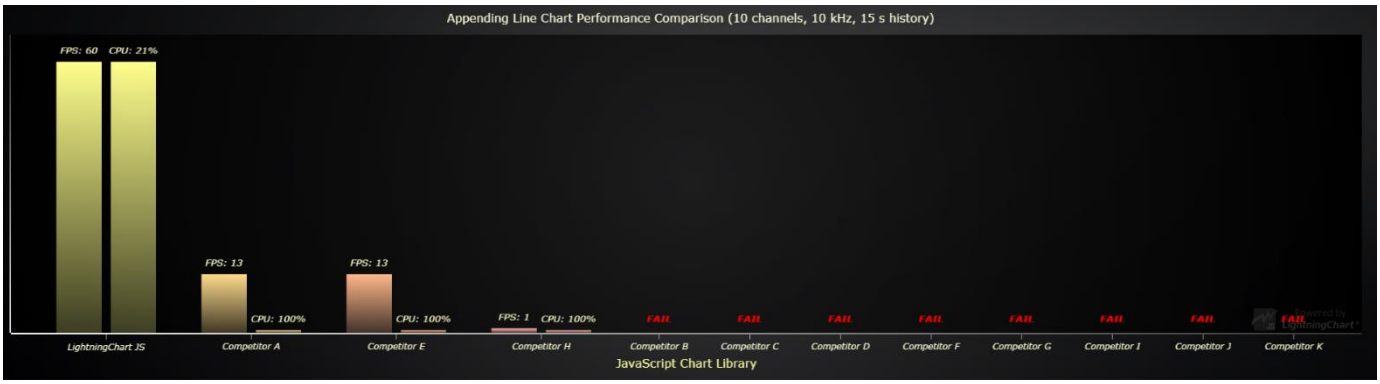
** Even with minimal data amounts, chart was stuck in loading animation.

Appending line chart performance comparison breakdown

Performance in appending chart applications is measured same way as in refreshing applications. However, generally refresh rates are much more frequent, usually capped around 60 FPS when the application is performing well.

We have selected a single test from the set of appending performance tests. This test was the same for each library and it highlights the performance differences most effectively. Here are the results of appending test with 10 channels, 10000 data points added every second (for each channel) and 15 seconds of displayed data history.

JavaScript Chart Library	Refresh rate (FPS)	CPU Usage (%)
LightningChart JS	60	21 %
Hardware accelerated competitor A	13	100 %
Competitor E	13	100 %
Competitor H	1	100 %
Competitor F	Fail	Fail
Competitor B	Fail	Fail
Competitor J	Fail	Fail
Competitor G	Fail	Fail
Competitor C	Fail	Fail
Hardware accelerated competitor D	Fail	Fail
Competitor I	Fail	Fail
Competitor K	Fail	Fail



As you can see from the amount of chart libraries which failed this test scenario, this is not a lightweight test.

When compared to other hardware accelerated charts, LightningChart JS could process on average 19.8x more data while using 4.2x less CPU power and refreshing 4 times faster. Combining these factors, **LightningChart JS was 332.6 times more powerful than other hardware accelerated charts**.

JavaScript Chart Library	Incoming data per second	Refresh rate (FPS) *	CPU usage (%)
LightningChart JS	1 million	**59.3**	**23.7 %**
Hardware accelerated competitor A	100 thousand	13.1	100.0 %
Hardware accelerated competitor D	1 thousand	16.9	98.7 %

When compared to non-hardware accelerated charts, LightningChart JS could process on average 18000x more data while using 4.1x less CPU power and refreshing 7 times faster. Combining these factors, **LightningChart JS was 516600 times more powerful than non-hardware accelerated charts**.

JavaScript Chart Library	Incoming data per second	Refresh rate (FPS) *	CPU usage (%)
LightningChart JS	1 million	**59.3**	**23.7 %**
Competitor E	100 thousand	13.2	100.0 %
Competitor H	10 thousand	10.6	100.0 %
Competitor F	10 thousand	10.1	100.0 %
Competitor C	10 thousand	9.4	100.0 %
Competitor B	10 thousand	4.3	100.0 %
Competitor G	10 thousand	8.5	100.0 %
Competitor J	10 thousand	4.4	81.9 %
Competitor I	1 thousand	10.8	100.0 %
Competitor K	1 thousand	4.3	100.0 %

* Average result of Google Chrome and Mozilla Firefox.

LightningChart JS Line Chart Capabilities

As you might know, LightningChart JS utilizes hardware acceleration for its graphics. This results in three very particular performance properties:

- **Low CPU usage**
  - As you can see from both highlighted real-time performance scenarios, LightningChart JS is extremely efficient on CPU usage with stark contrast to other chart libraries.
- **High refresh rate**
  - In all highlighted real-time performance scenarios, LightningChart JS refreshes with the maximum required display rate.
- **Hardware scaling**
  - Perhaps something which is not talked about enough; hardware acceleration enables utilizing the power of device graphics processing units (GPU). As a result of this, LightningChart JS performance skyrockets when powerful hardware is used.

It is worth noting, that this is not as simple as "if something is hardware accelerated then it must perform well". There are large differences even between performance of hardware accelerated web charts.

Let's see what happens when LightningChart JS is used with a powerful machine ...

We performed a separate test iteration with a more powerful PC (Ryzen 9 5900X, 64GB RAM, RTX 3080) to see what the maximum capability of LightningChart JS Line charts is. Here's the results!

Static line chart

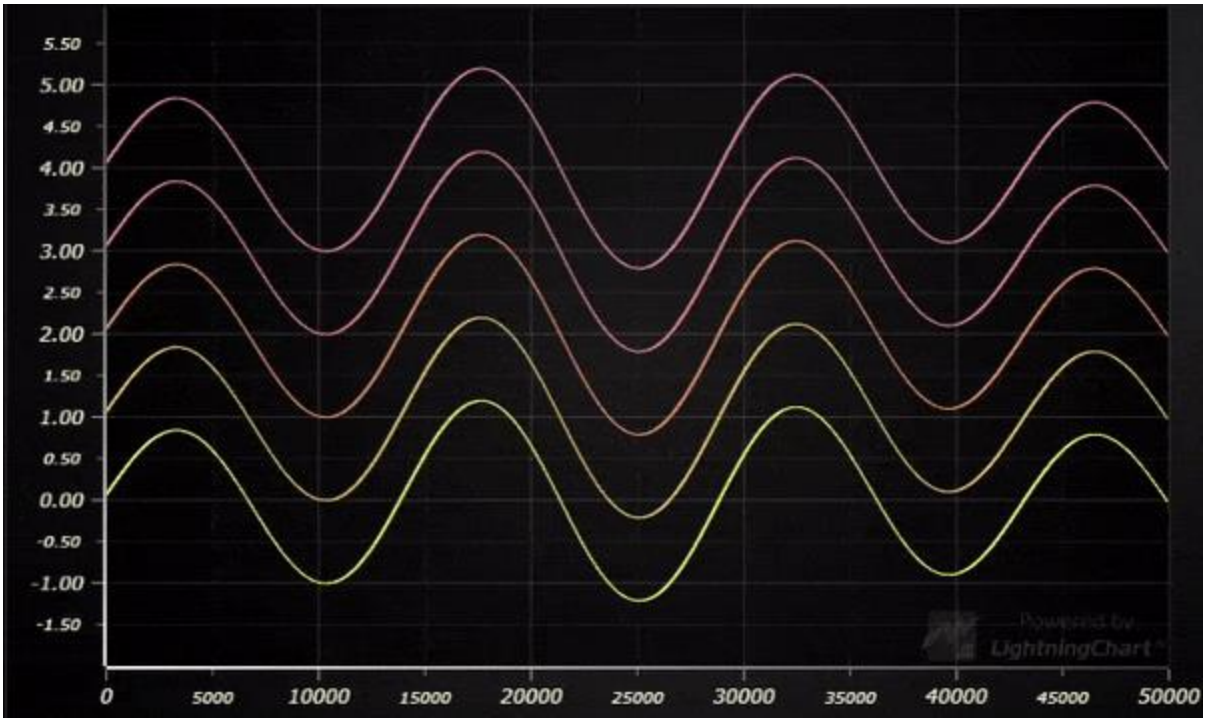
- Maximum data set size: **500 million data points**
- Massive line chart with 100 million data points can be loaded in 6.5 seconds!

Refreshing line chart

- **LightningChart JS officially enables real-time refreshing line data visualization**. From the performance results of older data visualization tools, it can be seen that they are simply not efficient enough with CPU usage to allow this kind of applications. Here is one performance test result we'd like to highlight:

JavaScript chart library	Refresh rate (Hz)	Total data points per refresh	Achieved refresh rate (FPS)	CPU usage (%)
LightningChart JS	10	2 million	**10.0**	**31.0%**

In this test, a considerably large line chart data set is refreshed 10 times per second. Note, the CPU usage from LightningChart JS: **31.0 %**. This leaves plenty of power for the rest of the web page as well as something often forgotten before it is a problem: transferring the data to the data visualization application, as well as possible data analysis computations.



Appending line chart

- **LightningChart JS officially enables real-time appending line data visualization**. From the performance results of older data visualization tools, it can be seen that they are simply not efficient enough with CPU usage to allow this kind of applications.

Why is this?

Most importantly, this is due to design decisions - there is large variety in the data management methods of different JavaScript charts. In an appending line chart, we identify three main methods of interacting with data:

1. Specifying data set.
    - This is where data visualization starts, you supply the chart library with a data set and you get the visualization in response.
2. Appending new data on top of previously added data.
    - **A must have feature** for appending data applications, this allows efficient data updates when old data is not modified changed and just 1 or couple samples are added.
    - Very few JavaScript charts support this feature, which shows quite clearly in their appending performance.
3. Removing old data that is out of view.
    - In real-time monitoring applications it is quite common that applications can run for long times or even indefinitely. For this reason, it is **vital** to remove old data periodically.
    - We didn't find any JavaScript chart other than LightningChart JS which supports this out of the box. All responsibility of cleaning data is on the user, which generally results in low performance and a lot of extra work.

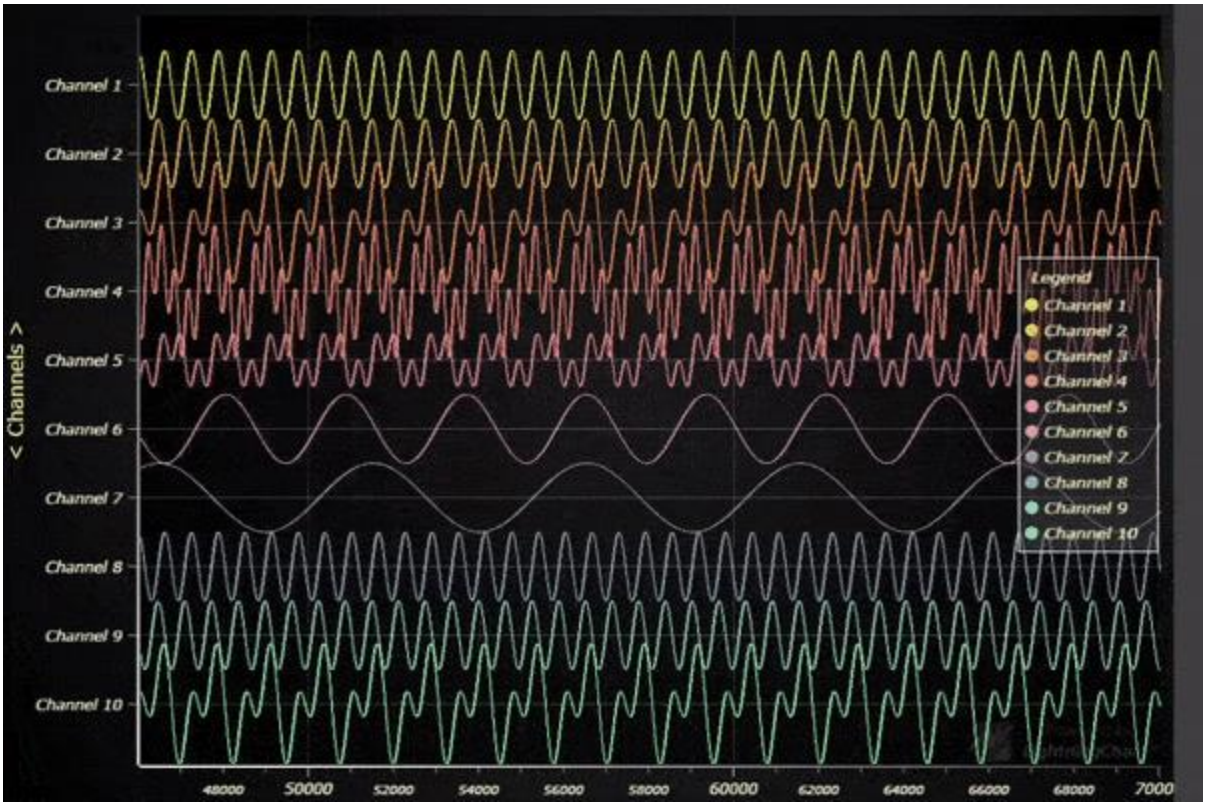
How does LightningChart resolve this issue?

From the start, LightningChart JS was designed to work in all real-time applications. For this reason, we our line series features set handles all the above-mentioned processes internally, while user only has to push in new samples to append.

...and here is how it performs with a fast machine:

JavaScript chart library	Channels count	Input frequency	Total new data points per second	Achieved refresh rate (FPS)	CPU usage (%)
LightningChart JS	20	200 kHz	4 million	128	50 %

This is a seriously heavy application, with high number of channels and extreme input frequency. In practice, this should cover any realistic need for real-time line data visualization applications, which are usually limited by input rate and total displayed data points count.



Errors in data visualization

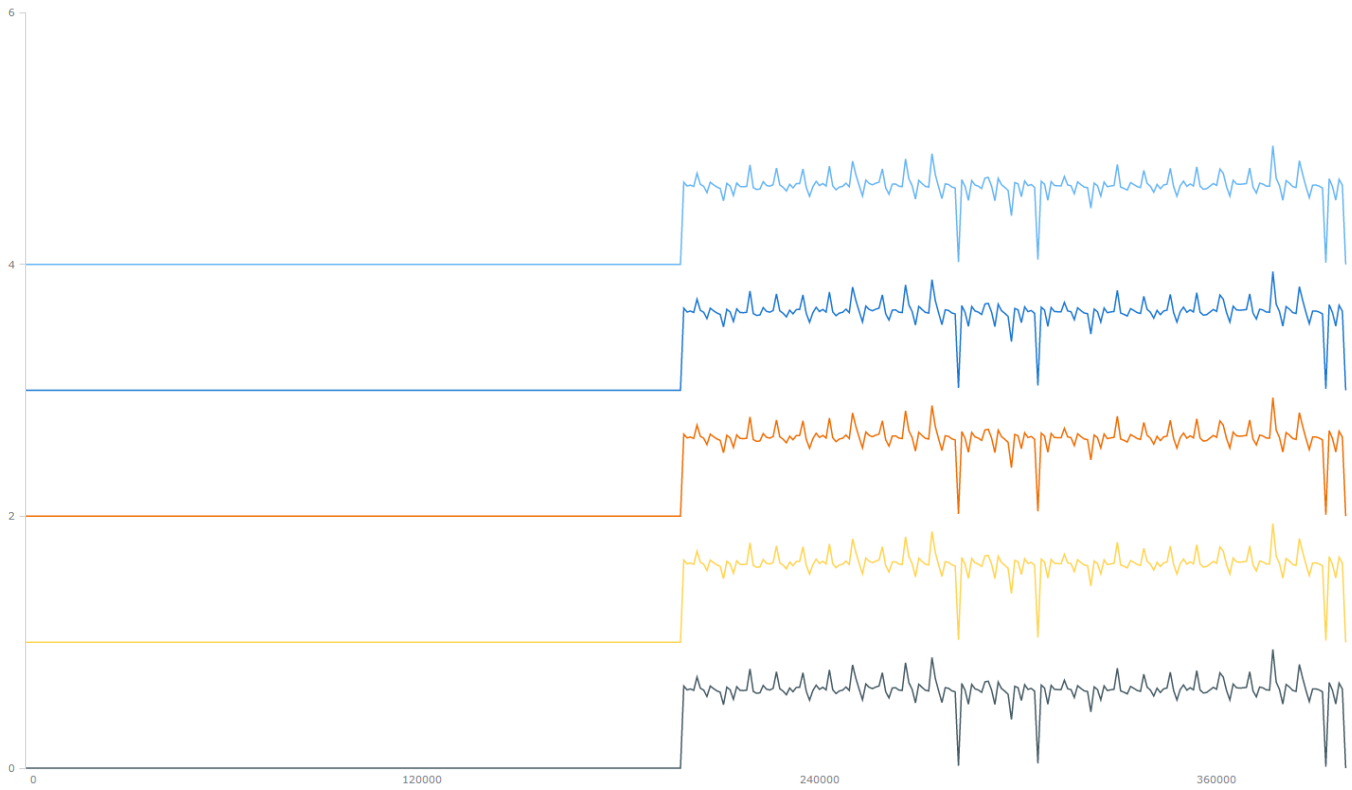
Chart library performance is important, but producing correct visualizations is even more important. This section contains some cases where **incorrect** or *unexpected* results were identified.

Down sampling

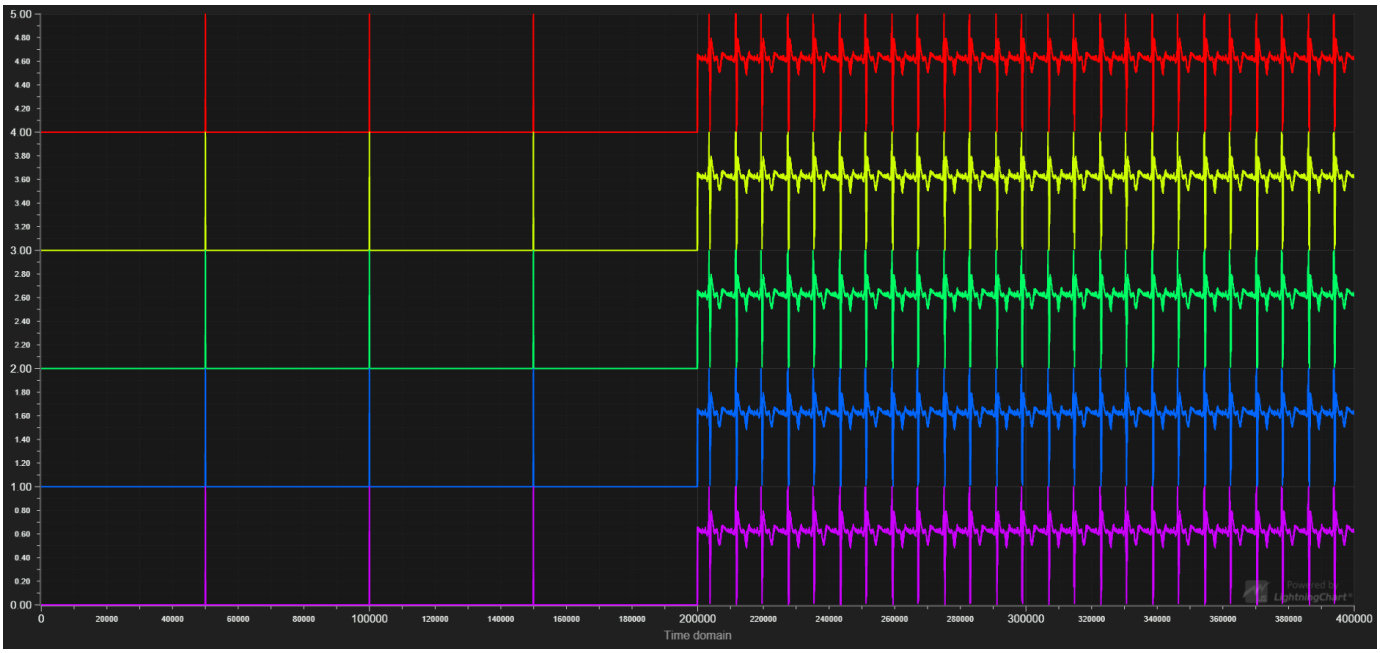
Some chart libraries are known to utilize internal down sampling of input data. While these yields increased performance, it produces incorrect visualization which is unacceptable in any realistic application.

The following competitors have been tested and proven to utilize down sampling:

- Competitor C

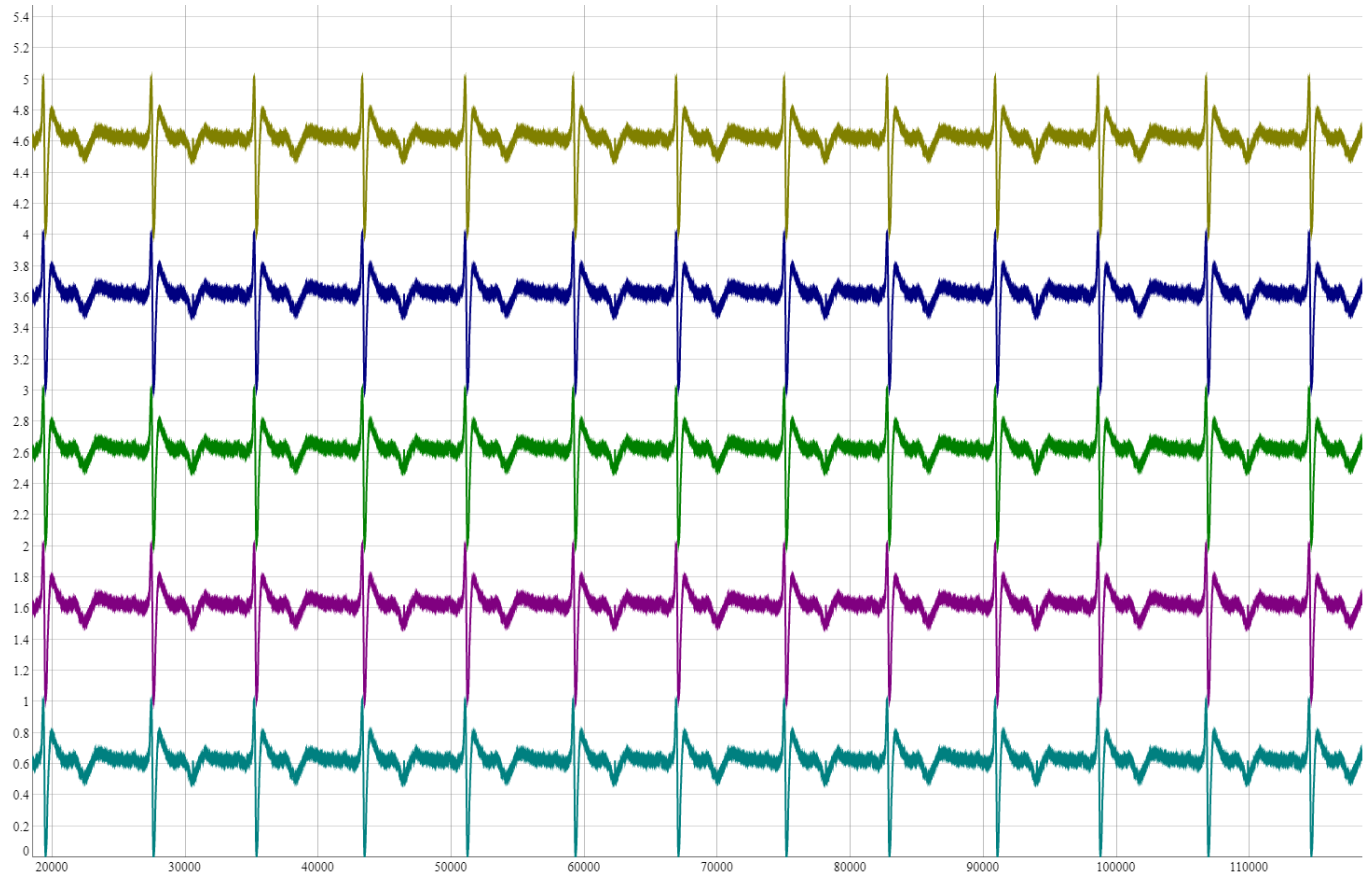


Same data visualized *correctly* with LightningChart® JS:



Other errors

With extremely dense data (100 µs resolution), competitor G produces incorrect visualization (curve looks like it is thicker than it should, or like there is a lot of *noise*). In this case, the Y value can't be accurately identified. Additionally, the visualized Y min/max range is out of the input data bounds (max Y is obviously drawn higher than 5.0).



Same data visualized *correctly* with LightningChart® JS:

What is the fastest JavaScript chart?

In conclusion: LightningChart JS v.3.3.0 has the smallest initial rendering time, it runs with a very compact memory footprint with the lowest CPU overhead, and the fastest response to mouse interactions.

LightningChart JS reaches the highest usable data points in all the performed tests compared to other charting libraries.

Also, note that some competitors use down sampling as an option to increase FPS. Down sampling really should not be used or tolerated by any software supplier making realistic and believable applications. Without down sampling, these performance results would have been much worse for some.

Remarkable observations:

On a mid-level device,

- LightningChart JS **static line charts** are **65.7x faster** than **non-hardware-accelerated** libraries and **7.5x faster** than **hardware-accelerated** libraries.
- On average, LightningChart JS **refreshing line charts** are **14.2x faster** than **non-hardware accelerated** libraries and **9.1x** faster than **hardware-accelerated** libraries.
- On average, LightningChart JS **appending line charts** are **516,600x faster** than **non-hardware accelerated** libraries and **332.6x** faster than **hardware-accelerated** libraries.

About LightningChart® JS

LightningChart® is registered trademark by Arction Ltd, a pioneer in high-performance charting, who introduced the fastest, GPU accelerated charts, already in 2009 for Microsoft .NET technologies. Before that, and ever since, the LightningChart® team has studied different technologies, prototyped, researched, innovated new algorithms, which are now part of LightningChart® product lines, to produce the absolute best performance for those advanced applications that really need it.

LightningChart® JS product line was released in 2019 and development is full-time by a large team. LightningChart® team is ready to help!