LightningChart®

JavaScript Charts Performance Comparison / Surface Grid Charts

Test date November 26th, 2021

Foreword

This is a performance test / comparison of JavaScript charts. The tests are focusing on surface grid charts' performance in different scenarios – visualizing static data sets, refreshing datasets, and appending data sources.

The charts selected in this test, are the major manufacturers who claim their charts to be high-performance oriented or the fastest, and some open-source libraries. There are also other charts available, which are either end-of-life, not supported anymore, or simply don't work. They were excluded.

We are confident we have selected all the fastest charts in the comparison, and if we didn't, inform us to get it added in this test. The surface grid charts in these tests should cover almost all application fields for surface grid charts.

Test procedure

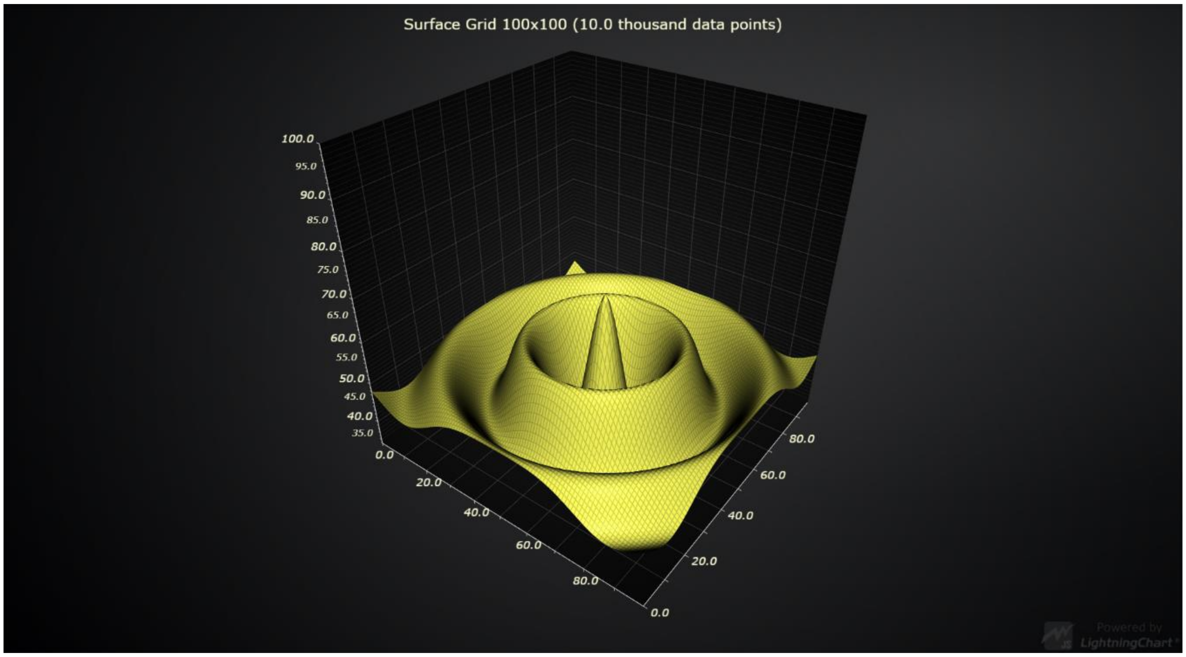
The test project is published as open-source project in GitHub.

Surface grid charts are used for visualizing of at least 3-dimensional data which exists on a plane (usually referred to as X and Z axes). The best way to understand this scenario is visualization of geospatial information, which has latitude (X), longitude (Z) and height (Y) values.

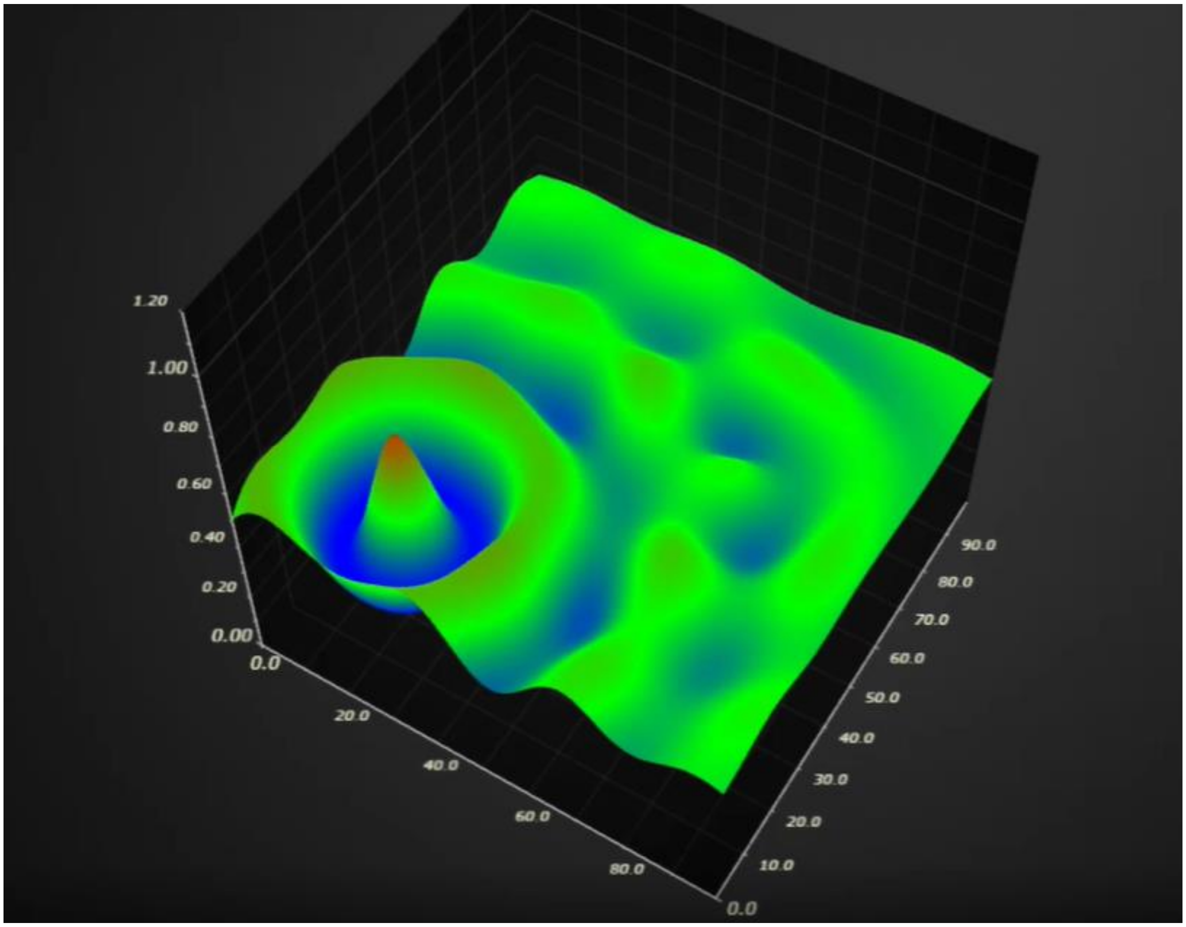
Additionally, surface grid charts are also used in 4-dimensional data visualization by coloring the surface dynamically based on a 4th data dimension.

For testing performance in different types of applications, we have identified 3 different application types of surface grid charts:

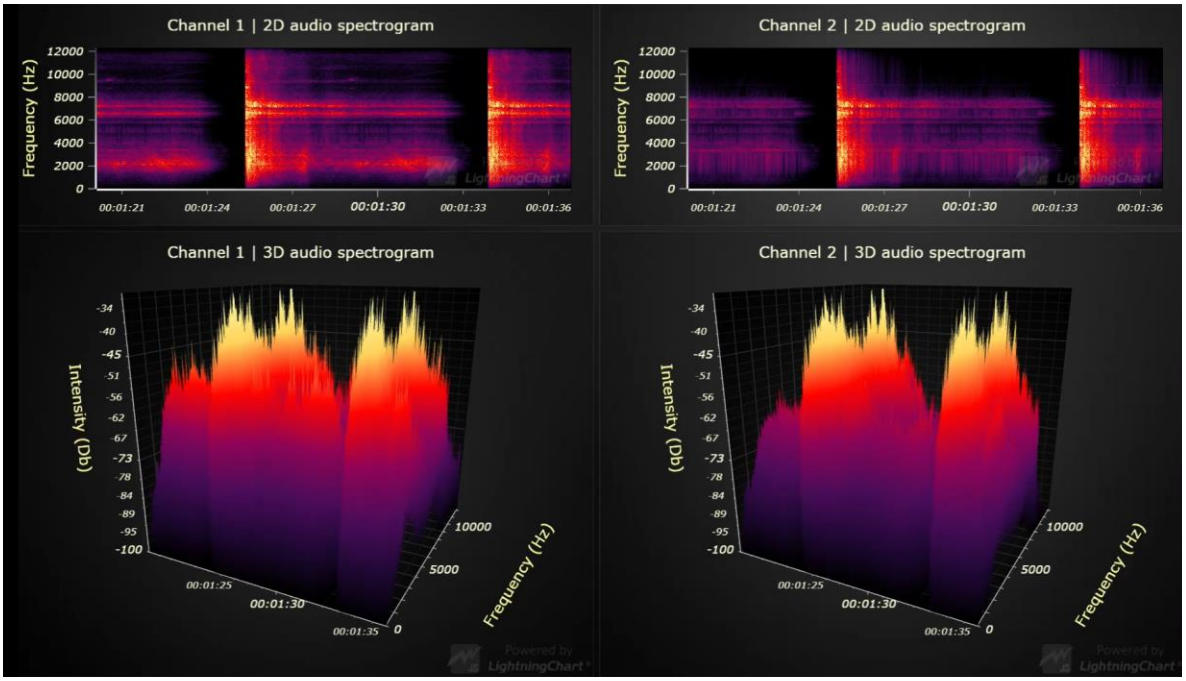
1. Static surface chart. A height map data set is loaded and rendered as surface chart.



2. Refreshing surface chart. In this case, the data is dynamic changing every so often (refresh rate). Used in real-time monitoring / analysis of geospatial data.



3. Appending surface chart. Also, dynamic data, but in this case the previous data is not cleared, instead just shifted out as new data is pushed in. Used in audio monitoring and analysis (spectrograms), for example.



Test hardware setups

These were measured on 24.11.2021, with an average office PC (Intel Core i7-7700K, 16 GB RAM, AMD Radeon R9 380).

JavaScript chart performance in surface chart applications is measured by gathering different performance metrics from a collection of surface dimensions.

Surface dimensions are specified by the number of columns and rows, for example "100x100" (10 000 data points).

Test chart libraries (in no particular order)

- LightningChart® JS v.3.3.0
- ECharts V.5.2.2
- SciCharts JS v.2.0.2115
- Plotly JS v.2.4.2

Results (1/3)

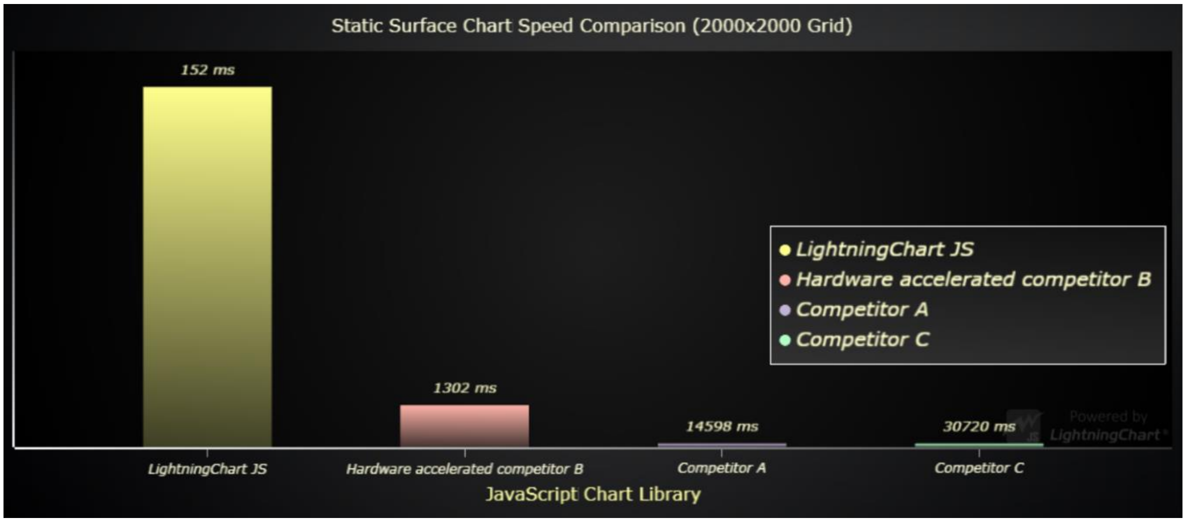
Static performance comparison breakdown

In static data visualization, the most important measurable performance attribute is how fast the chart is displayed to the user. We have a selected a single test from the set of static performance tests that were run for each included chart library.

This test is the same for each library and it highlights the performance differences most effectively.

Here are the results of static surface grid chart test with 2000x2000 data points.

JavaScript Chart Library	**Loading speed (milliseconds)**
LightningChart JS	152
Hardware accelerated competitor B	1302
Competitor A with no hardware acceleration	14598
Competitor C with no hardware acceleration	30720



On average LightningChart JS was ~60x faster than other charts. However, direct comparison can't be justified in this manner since LightningChart JS reaches much larger data sets than other charts.

Surface grid size	**Non hardware accelerated charts average speed**	**Hardware accelerated charts speed**	**LightningChart speed**
100x100	517 ms	331 ms	105 ms
1000x1000	4583 ms	584 ms	125 ms
2000x2000	22659 ms	1302 ms	152 ms
4000x4000	Fail	4838 ms	232 ms
6000x6000	Fail	9501 ms	374 ms
8000x8000	Fail	Fail	614 ms
10000x10000	Fail	Fail	829 ms
12000x12000	Fail	Fail	1260 ms

This is a good place to explain what does the "loading speed" measurement include. You might run into various claims of JavaScript loading speed on 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.

Results (2/3)

Refreshing 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.

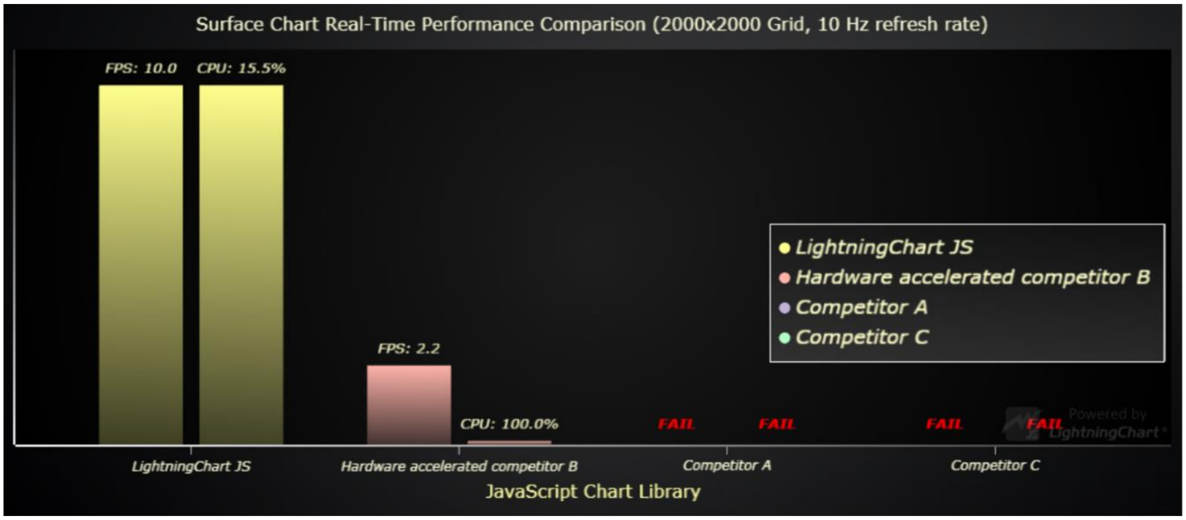
If your web page has a chart component which uses 100% of CPU, you can say goodbye to your good user experience.

We have selected a single test from the set of refreshing performance tests that were run for each included chart library. This test is the same for each library and it highlights the performance differences most effectively.

Here are the results of refreshing (refresh rate = 10 Hz) surface grid chart test with 2000x2000 data points.

JavaScript Chart Library	**Actual refresh rate /s**	**CPU Usage (%)**
LightningChart JS	10.0	15.5
Hardware accelerated competitor B	2.2	100.0
Competitor A with no hardware acceleration	Fail	Fail
Competitor C with no hardware acceleration	Fail	Fail

Below is a bar chart visualization of this same results table.



On average, LightningChart JS processed data 430x faster than non-hardware accelerated charts and 18.2x faster than other hardware accelerated charts.

JavaScript Chart Library	**Max data process speed**	**Surface grid size**	**Achieved refresh rate**
LightningChart JS	160 M/s	4000x4000	10 Hz
Hardware accelerated competitor B	8.8 M/s	2000x2000	2.2 Hz
Competitor A with no hardware acceleration	600 k/s	500x500	10 Hz
Competitor C with no hardware acceleration	148 k/s	200x200	3.7 Hz

Results (3/3)

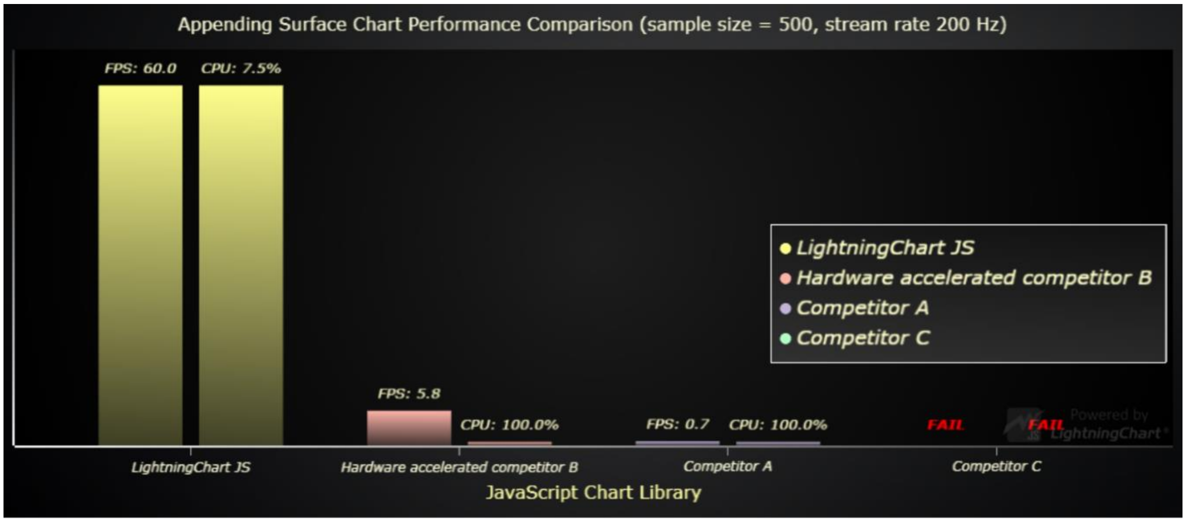
Appending performance comparison breakdown

We have a selected a single test from the set of appending performance tests that were run for each included chart library. This test is the same for each library and it highlights the performance differences most effectively.

Here are the results of appending surface grid chart test with sample size 500, samples added per second 200 and sample history 10 seconds.

JavaScript Chart Library	Refresh rate (FPS)	CPU Usage (%)
LightningChart JS	60.0	7.5
Hardware accelerated competitor B	5.8	100.0
Competitor A with no hardware acceleration	0.7	100.0
Competitor C with no hardware acceleration	Fail	Fail

Below is a bar chart visualization of this same results table.



On average, LightningChart JS could manage appending applications with **1000x more data** than non-hardware accelerated charts and **20x more data** than other hardware accelerated charts, while requiring **significantly less CPU power**.

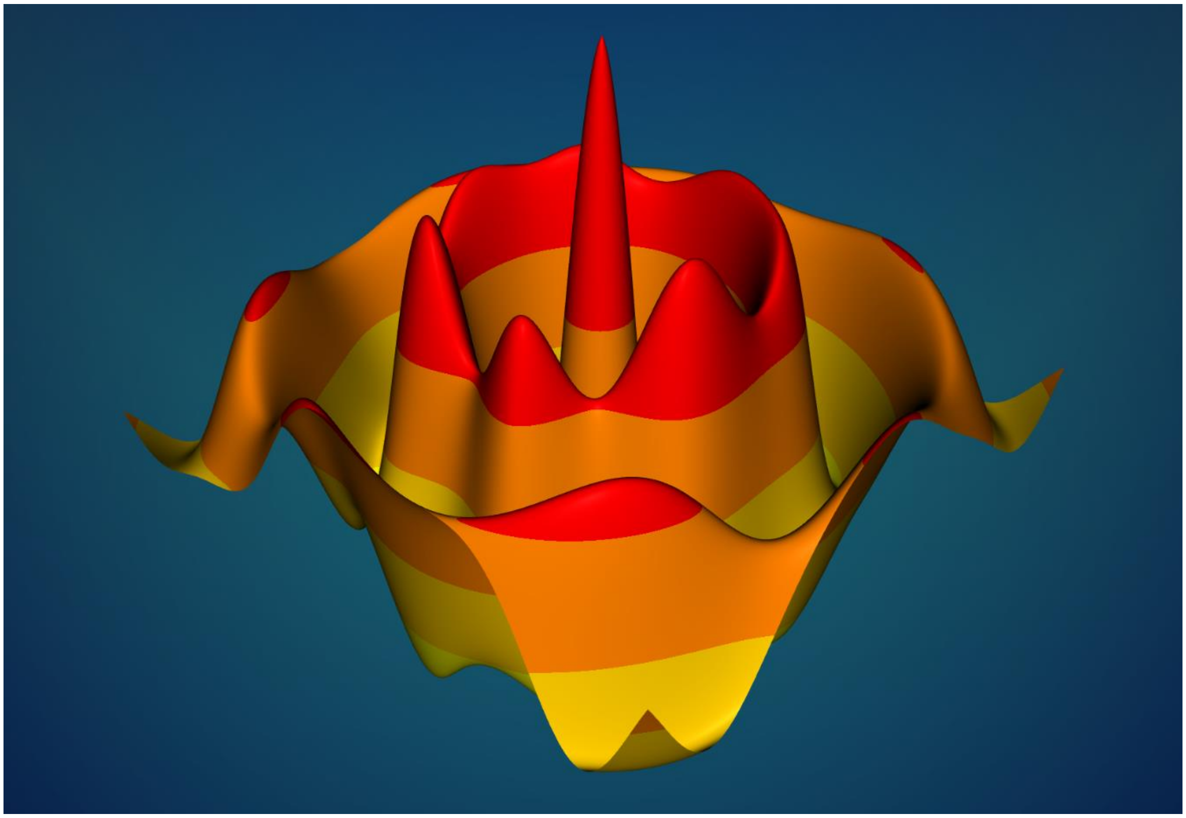
JavaScript chart library	**Heaviest test while keeping FPS > 10**	**Incoming data points per second**	**CPU usage (%)**
LightningChart JS	Sample size: 1000, Stream rate: 200 Hz	200 000	5.3%
Hardware accelerated competitor B	Sample size: 100, Stream rate: 100 Hz	10 000	100.0%
Competitor A with no hardware acceleration	Sample size: 100, Stream rate: 10 Hz	1 000	79.0%
Competitor C with no hardware acceleration	Sample size: 100, Stream rate: 10 Hz	1 000	100.0%

Additional Test

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 Surface charts is. Here are the results:

Static surface chart

- Maximum data set size: **2 BILLION data points** (45000x45000)
- Massive 10000x10000 surface grid can be loaded in less than a second! (768 ms)
    - This translates to processing ~130 million data points in 1 second.



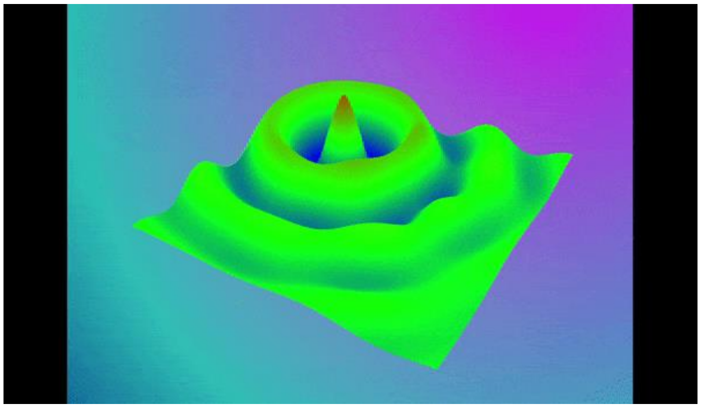
Refreshing surface chart

- LightningChart JS officially enables real-time refreshing surface 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)**	**Surface grid dimensions**	**Total data points per refresh**	**Achieved refresh rate (FPS)**	**CPU usage (%)**
LightningChart JS	60	1000x1000	1 million	60.0	16.0%

In this test, a surface data set is refreshed 60 times per second. This is the most common maximum refresh rate of computer monitors, thus a very commonly used refresh rate in monitoring solutions.

Note, the CPU usage from LightningChart JS: 16.0 %. 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 surface chart

- LightningChart JS officially enables real-time appending surface 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. All other chart solutions that we tested only allowed following actions:

- Create surface chart with X data set.
- Update existing surface chart with X data set.

However, this is not applicable to appending surface charts because of several reasons:

1. User is responsible for appending data and shifting old data out.
    - This means that actually users are implementing a significant part of the data processing.
2. Data update is not optimized.
    - Even if only one sample is added to the surface, it results in the entire chart being updated as if the whole data set was changed.
    - This will NEVER perform on an acceptable level in real-time applications.

How does LightningChart resolve this issue?

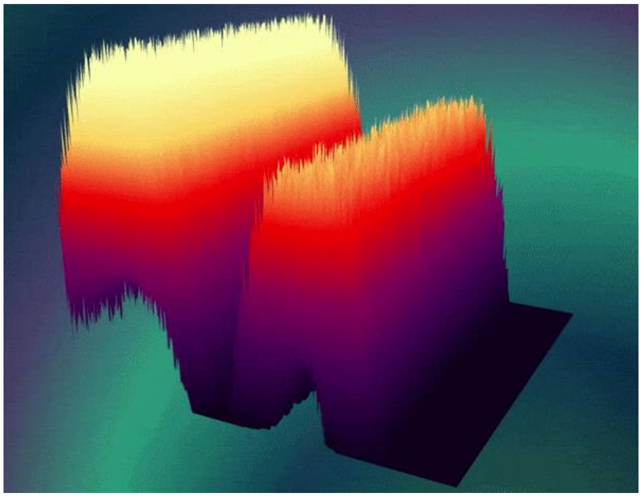
From the start, LightningChart JS was designed to work in all real-time applications. For this reason, we have a dedicated surface chart feature, which handles all the above-mentioned processes internally, while user only has to push in new samples to append.

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

JavaScript chart library	**Surface grid dimensions**	**New data points per second**	**Achieved refresh rate (FPS)**	**CPU usage (%)**
LightningChart JS	2000x1000	200 thousand	55.0	2.5%

This is an extremely heavy application, with each sample having 2000 data values and displaying time domain history from 10 seconds with 100 new samples added per second.

In practice, this should cover any realistic need for 3D spectrogram data visualization applications, which are usually limited by sample size and refresh rate.



Conclusion – Which is the fastest JavaScript chart?

Different chart charting libraries have their own strengths and selling points, but LightningChart's strength definitely is the exceptional rendering performance, allowing to build very advanced and data-intensive applications, visualizing heat map charts in just about any type of application.

After performing these intensive tests on three types of JS surface grid charts, it was demonstrated that LightningChart is the fastest hardware-accelerate charting library as is able to:

- keep a low CPU usage during all tests at any extremely large datasets.
- maintain the highest refresh rate.
- boost its performance level, even more, when carrying out the tests on a high-end device.

Remarkable observations:

On a mid-level device,

- LightningChart JS static **surface charts** are **60x faster** than other libraries
- On average, LightningChart JS refreshing **surface charts** are **430x faster** than non-hardware accelerated libraries and 18.2x faster than hardware-accelerated libraries.
- For appending **surface charts**, LightningChart JS can manage, on average, **1000x more data than non-hardware accelerated charts** whereas it can handle **20x more data than hardware-accelerated charts**.

On a high-level device,

- LightningChart JS static **surface charts** can **visualize 2 billion data points** and **load 130 million data points in** only **768 milliseconds**!
- For refreshing **surface charts**, LightningChart JS can **handle 1 million data points per refresh** at **60 FPS** and **only 16% CPU usage**.
- Appending **surface charts** support **dimensions of 2 million data points** at **55 FPS** and **only 2.5% CPU** is **required**.

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, Ultimate Data Visualization Solutions 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!