

Design and Evaluation of WebGL-Based Heat Map Visualization for Big Point Data

Jan Ježek, Karel Jedlička, Tomáš Mildorf, Jáchym Kellar
and Daniel Beran

Abstract Depicting a large number of points on a map may lead to overplotting and to a visual clutter. One of the widely accepted visualization methods that provides a good overview of a spatial distribution of a large number of points is a heat map. Interactions for efficient data exploration, such as zooming, filtering or parameters' adjustments, are highly demanding on the heat map construction. This is true especially in the case of big data. In this paper, we focus on a novel approach of estimating the kernel density and heat map visualization by utilizing a graphical processing unit. We designed a web-based JavaScript library dedicated to heat map rendering and user interactions through WebGL. The designed library enables to render a heat map as an overlay over a background map provided by a third party API (e.g. Open Layers) in the scope of milliseconds, even for data size exceeding one million points. In order to validate our approach, we designed a demo application visualizing a car accident dataset in the Great Britain. The described solution proves fast rendering times (below 100 ms) even for dataset up to 1.5 million points and outperforms mainstream systems such as the Google Maps API, Leaflet heat map plugin or ESRI's ArcGIS online. Such performance enables interactive adjustments of the heat map parameters required by various domain experts. The described implementation is a part of the WebGLayer open source information visualization library.

J. Ježek (✉) · K. Jedlička · T. Mildorf · J. Kellar · D. Beran
Section of Geomatics, Department of Mathematics, Faculty of Applied Sciences,
University of West Bohemia, Univerzitní 8, 306 14 Plzeň, Czech Republic
e-mail: jezekjan@kma.zcu.cz

K. Jedlička
e-mail: smrcek@kma.zcu.cz

T. Mildorf
e-mail: mildorf@kma.zcu.cz

J. Kellar
e-mail: kellar@students.zcu.cz

D. Beran
e-mail: dberan@students.zcu.cz

Keywords Kernel density map · Heat map · WebGL · GPU · Visualization

1 Introduction

Depicting a point symbol on a map is the most common approach to display a location of a point feature. However, when considering a big amount of point features, such technique leads to over-plotting that may overwhelm users' perception and cognitive capacities. A relevant visual approach that helps to interpret spatial distribution of point data is the kernel density map (sometimes called heat map). This colored isarithmic map can be used to depict the density of spatial features. Efficient exploration of point data through the density map often requires interactions such as zooming or heat map parameters adjustment and filtering. These interactions demand low latency responsiveness even in the case of large dataset. However, calculation of such a map might be too complex to achieve this without advanced techniques.

In this paper we propose a system for an efficient kernel density estimation and heat map generation for the web by utilizing hardware acceleration through WebGL. We designed a system that enables visualization of a kernel density map for a dataset of up to 1.5 million points in a time that enables interactive frame rate. This Web-based system enables to display a heat map on top of a background map managed by a third party mapping library (e.g. Open Layers, Leaflet or Google Maps). Such speed enables fast interactions for zooming or calculation parameters adjustments even for the specified size of data.

In this paper, a detailed description of the kernel density map creation through hardware acceleration is given and several implementation considerations are discussed. For the demonstration purposes, a visualization application for up to 1.5 million points (traffic accidents) has been designed. We compared the solution with relevant systems such as the GoogleMap heat map API, Leaflet heat map plugin or ArcGIS Online heat map function. Furthermore we collected feedback for the effectiveness and usefulness of provided heat map interactions by domain experts (traffic engineers).

The outline of this paper is as follows. Section 2 sheds light on theoretical background related to density estimation as well as related mapping systems. Section 3 defines a basic methodology and terms for kernel density estimation used in our approach. Section 4 describes the details about the proposed GPU-based implementation. Section 5 shows the performance results and compares our approach with existing systems. This chapter also summarizes the user feedback. Then the conclusion is given.

2 Related Work

Visualization of big amount of geospatial point data is a topic tightly connected to various research areas. The number of point features that can be visualized on the screen is limited by the display size and resolution as well as by human perception abilities. In order to reduce the data size, there are two main approaches: (1) aggregate and sample the data, where data are grouped based on their spatial or semantic proximity to reasonable number of clusters (Fig. 1c), or (2) partition the space to disjoint areas and aggregate the data located in each area. These areas can be defined by spatial semantics (e.g. by province boundaries as depicted in Fig. 1a), or artificially by regularly dividing the space into cells (Fig. 1b).

The most relevant spatial clustering methods widely accepted in GIS were summarized by Fotheringham and Rogerson (1994). The general problem of spatial point pattern analysis is described by O’Sullivan and Unwin (2014), where the methods are divided to distance-based and density-based. The cartographic aspects and relevant visualization techniques are given by Slocum (2008). Clusters are most often visualized as a point symbol map where the amount of data is encoded to a visual variable (e.g. to the size of the symbol). Particular applications of these methods alongside with their advantages and disadvantages for big point data are further discussed by Kuo et al. (2006) and Skupin (2004).

In addition to clustering techniques that represent data as discrete values, there is a density map approach that represents the point distribution as a smooth continuous surface (Fig. 1d). The simple point density estimation method works on a predefined grid. Each cell of the grid has a specified circular neighbourhood used to calculate the cell density value as the ratio of the number of features that are located within the search area to the size of the area (Silverman 1986). The other approach

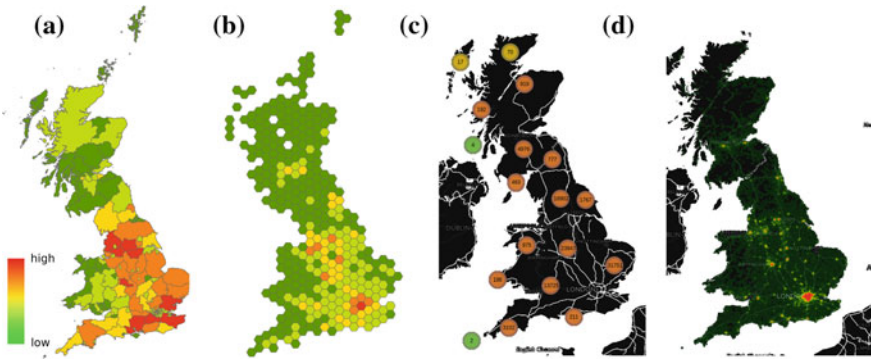


Fig. 1 Example of point density visualization through data reduction based on space partitioning (a, b), clustering (c) and the heat mapping (d). The maps visualize the same dataset containing 100,000 car accidents in Great Britain

is the kernel density estimation. Instead of considering a circular neighbourhood around each cell, as used in the simple method, a circular neighbourhood around each spatial feature is drawn (the kernel). Then a mathematical equation is applied that goes from 1 at the position of the feature point to 0 at the neighbourhood boundary (Fotheringham et al. 2000). The final surface is given by summing the values of the kernels.

In order to visualize such a density surface, the isarithmic map approach can be used. Specifically a contour map, hypsometric tints (isopleth map) or a continuous-tone map (heat map) (Slocum 2008). The application examples of density estimation and its visualization can be found in various domains, e.g. road safety (see Wongsuphasawat et al. 2009; Anderson 2009; Dillingham et al. 2011), or a criminality mapping (Chainey et al. 2008). Considering the above mentioned applications, the most widely used technique for visualization of a large number of points in these domains is the kernel density estimation visualized as a continuous-tone map, which is commonly called a heat map. Dillingham et al. (2011) validates the heat map as an effective visual technique for road accident data. Furthermore, a strong advantage of heat maps over the clustering approach is its ability to preserve outliers (Liu et al. 2013). A cartographic heat map has its background in the information visualization (Infovis) domain, where the heat map is used in slightly different manner. The history of heat maps, as considered in Infovis, is summarized by Wilkinson and Friendly (2009).

The kernel density estimation and its visualization using a heat map might be an expensive process for big data. Therefore, an efficient system architecture and algorithm design is desired. One of the relevant speed-up techniques is the algorithm parallelism, enabled by a utility of the graphical processing unit (GPU) [as demonstrated by Srinivasan et al. (2010)]. The GPU-based approach proofs its benefits in various similar scenarios. For example Fekete and Plaisant (2002) use the GPU approach to produce a treemap of 1 million records. Liu et al. (2013), Lins et al. (2013) present a visualization of billions of records through linked views by using the GPU approach and Hennebohl et al. (2011) demonstrate the GPU-based kriging. However, a GPU-based heat map designed for the purpose of web mapping has not been investigated so far.

In order to achieve an immediate response of visualization interactions, a value of 100 ms response time is considered (Card et al. 1983). Even though the most recent research shows the benefits of using a heat map for various scenarios, widely accepted web GIS tools enable only limited scalability and the interactivity tends to be too slow for a heat map visualization. Mainstream GIS system as well as the geovisualization systems [e.g. the GeoViz toolkit (Hardisty and Robinson 2011), the CDV (Dykes 1998) or similar] usually do not allow to effectively visualize more than 100,000 points. Our work follows various successful systems focused on big data processing by using a graphical hardware.

3 Geological Characteristic

The proposed approach is based on the kernel density estimation (KDE). The KDE is a basic method for estimation of a density surface from irregularly distributed point data. The KDE involves placing a symmetrical surface over each point. Afterwards, an evaluation of the distance from the point to the reference location is done and then a sum of the value for all the surfaces for that reference location is calculated. This allows to place a kernel over each observation, and summing these individual kernels provides the density estimate for the distribution of the points. An estimation of the relative density around a location s can be obtained from the following:

$$\lambda_s = \sum_{i=1}^n \frac{1}{\pi r^2} k(d_{is}, r, m) \quad (1)$$

where λ_s is the density estimate at the location s ; n is the number of observations, r is the search radius (bandwidth) of the KDE (only points in this radius are used) and k is the weight function. k is a function of the point i at the distance d_{is} , the radius r and the weighting parameter m . An example of k is $k(d_{is}, r, m) : 1 - \frac{d_{is}^m}{r^m}$.

4 GPU-Based Approach

A kernel density estimation surface is usually represented in the form of a raster. In order to produce such a surface and visualize it through a heat map, these main steps must be performed: (1) project the data from an input coordinate system to the actual screen viewport, (2) render the kernel around each observation, (3) summarize the values for each pixel of the output raster and (4) use color gradient to calculate the final color for the visualization.

The complexity of the algorithm depends on the number of observations, the raster size and also the radius value. For interactive visualization that supports zooming, it might be important to enable the user to change the resolution as well as to adopt the radius or the weighting parameter (m) of the kernel function in the runtime. Furthermore, application of color gradient can be customized to be used just within the user defined interval.

In our approach we focus on the GPU-based technique. A GPU enables to process the input data in parallel by using thousands of cores of the GPU hardware. Modern browsers supports WebGL as an API for GPU-based programming. On the lower level, WebGL is designed to render geometry primitives (points, lines, triangles). WebGL uses shader programs that run on the GPU as parallel processes: vertex shader assign a screen position to the input data and fragment shader that computes a color of every pixel of a geometry primitive and writes that in the frame buffer as a pixel. During this process, the blending function evaluates how to

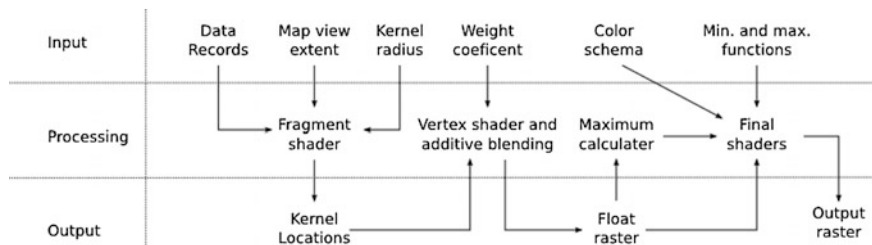


Fig. 2 Processing workflow schema

combine the new pixel color with an old color contained in the framebuffer. Due to the GPU architecture, the shaders run in parallel and allow efficient data processing on the one hand. On the other hand, the transmission from CPU to GPU is an expensive process that might slow down the performance.

In order to produce a heat map by using WebGL, we designed a workflow depicted in Fig. 2. The resulting heat map is considered as an overlay over user defined background map that supports zooming and panning. Such a background map is supposed to have a projected coordinate system and an actual geographic extent as well as dedicated viewport on the screen (map view extent) expressed by the height and width in pixels. The input data records are considered to be expressed in the same projected coordinate system. In order to display them into the viewport, only scaling and translation need to be applied. The resolution of the heat map is equal to the resolution of the current screen viewport. In our approach, all data are uploaded to the GPU during the initialization phase and the translation and scaling are performed according to the actual zoom level or panning operation within the vertex shader. This allows to transmit just information about the map extent from CPU to GPU during panning and zooming interactions.

In order to render the kernel around each observation, the circle area for every point is allocated according to the kernel radius. For each pixel within this kernel the value of the kernel function is calculated by the fragment shader and this value is assigned to the pixel (a float raster is used). Our implementation allows to set a weight coefficient (n) of the kernel function.

Afterwards, the WebGL enables to set the blending function that influences how newly calculated pixels should be combined with the pixels already stored in the framebuffer. In the case of a heat map, an additive blending is set to calculate the sum of the values for a particular pixel.

Afterwards, the resulting float raster is processed and the maximum value from all the pixels is withdrawn. For this purpose, a GPU technique based on generating series of mipmap of original texture is used (depicted as the maximum calculator in Fig. 2). This maximum is an input to specify the final range to be displayed by the heat map.

Finally, the produced density surface is processed once again by the WebGL shaders and a particular color schema is applied to produce the final heat map. For such a purpose, a user defined color scheme is set (e.g. green, yellow, red) and the

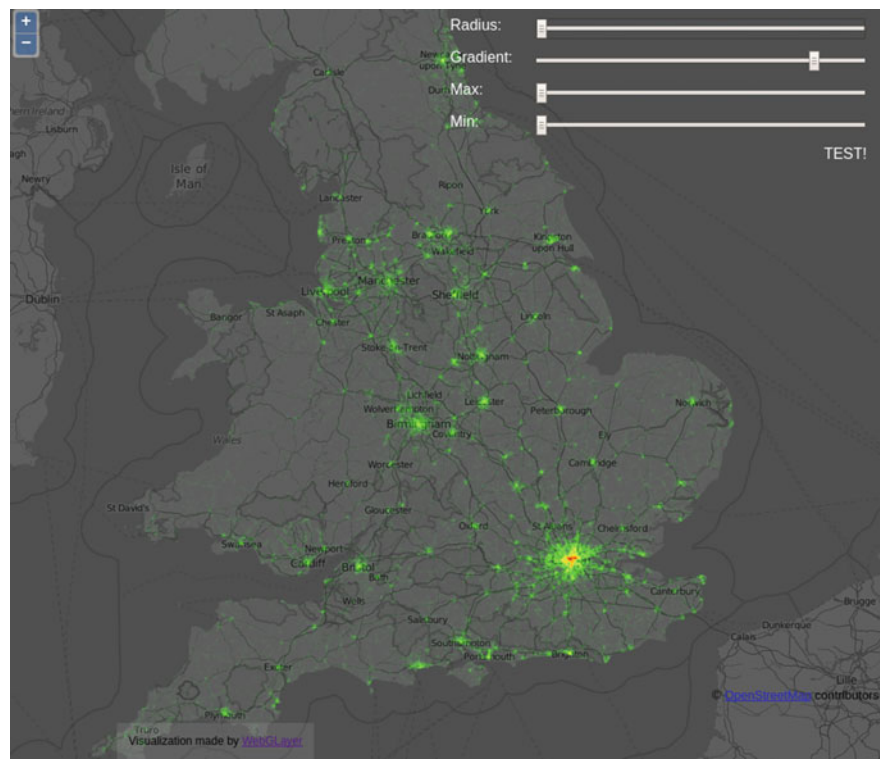


Fig. 3 GPU-based visualization of a heat map of 1,494,275 car accidents in Great Britain. Radius size 4 px

final value of the pixel is interpolated by using color gradient. One of the well-known and often used color schemas is the color gradient from green across yellow to red (e.g. a default schema in Google Maps Heat map API). It is useful to use different transparency for locations with low density and locations with high density so that the low density places do not overplot the background map while the high density places are less transparent. The final visualization of the car accident dataset by applying the mentioned color schema and transparency is depicted in Fig. 3.

The implementation of the described technique has been done as a part of the WebGLayer javascript library (see <http://jezekjan.github.io/webglayer>). The WebGLayer is JavaScript, WebGL based library for coordinated multiple views visualization. The implementation provides input variables including the radius, the kernel function coefficient, the range of minimum and maximum values of the kernel density estimation to be visualized through color gradients.

5 Comparison of the Developed GPU-Based Approach to Current Solutions

In order to validate our approach we implemented a demonstration application, a heat map of car accidents of the Great Britain. For this purpose, the road accident data from the data.gov.uk portal were used (see <https://data.gov.uk/dataset/road-accidents-safety-data/resource/626a5e18-be29-4fcf-b814-9c4edb4f3cc6>). The developed GPU-based solution was compared with three widely used techniques. The demo application is available online at: http://home.zcu.cz/~jezekjan/wgl_git/examples/heatmap/.

5.1 Methodology

In order to evaluate the performance and scalability of the solution, subsets of various sizes of the road accident dataset were prepared:

- Datasets of a size of 10,000, 15,000, 25,000 and 27,000 road accidents in the Birmingham metropolitan area. The datasets from 10,000 to 25,000 were diluted for the purposes of this test, the dataset of 28,000 points shows all road accidents in Birmingham between the years 2005 and 2014.
- Datasets of ranging from 100,000 to 1,494,275 points (the maximum is the size of the dataset) that represents road accidents in Great Britain in the same time period. The datasets were diluted for the purpose of this test and present always top-k elements sorted by a timestamp.

Afterwards, four applications displaying heat maps of these datasets were implemented. In addition to the GPU-based application, other three solutions were used: ArcGIS Online, Google Maps API and the Leaflet heat map plugin. For each technology, the rendering time needed to redraw the complete heat map was measured. The measurements were done with respect to the radius value that affects the rendering time in most cases. Other available libraries such as OpenLayers has not been evaluated in details as their performance seems to be similar with the mentioned solutions.

5.2 ArcGIS Online

ArcGIS Online (a web GIS platform that allows the user to use, create, and share maps) was used (see <http://doc.arcgis.com/en/arcgis-online/reference/what-is-agol.htm>) for two reasons. Firstly, ArcGIS Online is a widely spread GIS platform. Secondly, it offers a heat map visualization technique as an example of data driven

visualization (see <http://blogs.esri.com/esri/arcgis/2015/03/03/whats-new-arcgis-online-March-2015/> or heat map section at the page <http://doc.arcgis.com/en/arcgis-online/create-maps/change-style.htm>), ~heat map as an overlay on another data (e.g. background map). The following text shortly describes the technical challenges that were faced and the results that were achieved.

During uploading and setting up feature services from source layers, a limitation of ArcGIS Online was identified In ArcGIS Online, by default, the maximum number of records returned by the server is set to one thousand (see <http://support.esri.com/en/knowledgebase/techarticles/detail/42901>). This can be changed following this procedure (see <http://support.esri.com/fr/knowledgebase/techarticles/detail/44060>), however the technical article says, that: *“Increasing the value impacts performance if there is a large amount of data added to a web map. Hence, querying and displaying data from the web map may take a significant amount of time”*. This was experienced during the testing and it can be clearly seen from Table 1. To redraw the heat map the radius change was performed. e.g. when the radius was changed from 5 to 6 pixels, the redraw delay was measured and written into under the radius 6 px value (see Table 1). The value was measured for all the datasets.

When data were successfully loaded for visualization, experiments with changing the radius started, starting with the smallest dataset and keeping the zoom to the Birmingham area. The achieved results are mentioned in Table 1. Values for the datasets 10–27 k were acquired. The values of the 100,000 dataset measurement could not be taken into account because the rendering ended with this warning: *“The layer did not draw completely as there are too many features to display in this area of the map”*. The ArcGIS Online platform doesn’t provide an exact time measuring tools, therefore, a stopwatch was used. The web application used for tests is accessible online at: <http://arcg.is/1lmRQ3t>.

Table 1 Comparison of ArcGIS Online, Google Maps and GPU solution

Dataset	ArcGIS Online		Google Maps		Leaflet	GPU	
Number of points	Radius change		Radius change		Radius	Radius	
	5–10 px (s)	10–20 px (s)	5–10 px (s)	10–20 px (s)	2–20 px	5 px	20 px
10,000	0.6	0.9	<1	<1	0.066	<0.032	
15,000	0.7	1.3	<1	<1	0.084		
20,000	0.8	1.6	<1	<1	0.110		
28,000	0.9	2.1	<1	<1	0.139	0.032	0.037
100,000	Layer did not load properly		<1	<1	0.180	0.046	0.080
250,000			1.5	2.5	0.341	0.046	0.138
500,000			3	4	1.382	0.052	0.253
750,000			5	6	1.520	0.067	0.349
1,000,000			6.5	8	1.914	0.068	0.420
1,494,275			–	–	–	0.100	0.603

5.3 *Google Maps*

Another way to create a heat map is provided by the Google Maps Javascript API v3. It allows the user to use the heat map layer as an overlay on the top of the map. Data can be rendered either according to their coordinates or weight or a combination of both of them. The appearance of the heat map layer can be customized by changing such options as the radius of data points, the maximum intensity of the heat map, the color schema or the opacity (see <https://developers.google.com/maps/documentation/javascript/heatmaplayer>). Although the heat map layer can render a large number of point data, it may result in reduced performance. Again, a stopwatch was used, showing that the performance is better than in ArcGIS Online, but still losing interactivity and dynamics when a dataset of 100 k and more points is used. See Table 1 for all measured times. The map application can be found online at: <http://home.zcu.cz/~kellar/webglayer/heatmap.php>.

In addition to the heat map layer, which provides the client-side rendering of heat maps, the Google Maps Javascript API allows to use the server-side rendering via Fusion Tables. In this case, data are rendered on the server and sent to the user in the form of a raster tile overlay. This solution provides better performance for larger datasets, but obviously limits the interactions in runtime (e.g. to change the radius or maximum parameters is not possible). The Fusion Tables Layer allows to display first 100,000 rows of data in a table (see <https://developers.google.com/maps/documentation/javascript/fusiontableslayer#limits>). Due to these differences, the Fusion Tables Layer was not included in the test.

5.4 *Leaflet Heat Map Plugin*

The last considered solution is the Leaflet heat map plugin v0.2.0 provided by Vladimir Agafonkin (see <https://github.com/Leaflet/Leaflet.heat>). Leaflet as well as the heat map plugin are open source products used within the MapBox platform (see <https://www.mapbox.com>). The plugin is based on HTML and canvas 2D rendering technique and is inspired by more general tool heatmap.js (see <http://www.patrick-wied.at/static/heatmapjs>). Due to the open source nature of the project, it was possible to precisely measure the rendering time by modifying the source code. During the testing, no dependency between radius value and rendering time was observed. The measure time is an average time for various radius possibilities (from 2 to 22), while rendering the map 10 times for each radius. The map application can be found online at: <http://jachymkellar.github.io/heatmap/>.

5.5 Test Result and Comparison

The performance of the GPU-based solution was evaluated by measuring the time needed for the whole rendering workflow. As the testing methodology suggests, during the tests, rendering times for datasets of different size and of different radius of the kernel were measured. For the test the zoom level that displays the whole dataset (zoom level 9) was used and a resolution of the map was 1631×965 . All the tests were performed on a machine with the following configuration: Google Chrome 46.0.2490.80 (64-bit), Ubuntu Linux on Lenovo T430s, Intel Core i7-3520M CPU @ 2.90 GHz \times 4, RAM 3.6, graphics: NVIDIA, GF117M GeForce 610M/710M/820M/GT 620M/625M/630M/720M. The rendering workflow was repeated 50 times and the average time was calculated. The Google Chrome web browser was setup to enable all available graphics hardware accelerations (under chrome://gpu).

The results are available in Table 1. The results demonstrate that the ArcGIS Online solution is not suitable for big data. It slows down to more than a second when using larger radius on datasets around 15,000 points. The 100,000 dataset didn't even load properly.

Google Maps API can handle higher amount of data than ArcGIS Online, but it starts to have significant lags around 250,000 points (2.5 s for 20 px radius). Even though Google Maps API can work with 1,000,000 dataset, the rendering time is 26 s.

Considering all three concurrent solutions, the Leaflet heat map plugin was the fastest one. The time below 100 ms that is considered as suitable for interactive visualization was reached for dataset below 20,000 points. In contrast to all the other solutions, no dependency on the value of the radius was observed.

Comparing all four solutions, the proposed GPU-based approach gives the best results: the interactive response time was reached even for the case of visualizing the whole dataset (1,492,475 points) for the case of the radius below 6 pixels. The dependency on the radius size is significant, however, the use of big radius does not usually provide valuable visualization and leads to the loss of details as the peaks of the surface are smoothed in such a case.

The detailed results of the performance measurements of the GPU-based solution are depicted in Fig. 4. The GPU solution reacts immediately [$t < 100$ ms as defined in (Card et al. 1983)] for 100,000 dataset with 20 px radius or even 1,400,000 dataset with 6 px.

5.6 User Feedback

In addition to the performance and scalability testing, we asked for feedback from domain experts using the tested tools on a daily basis. We discussed the car accidents visualization with experienced traffic engineers from EDIP s.r.o to validate

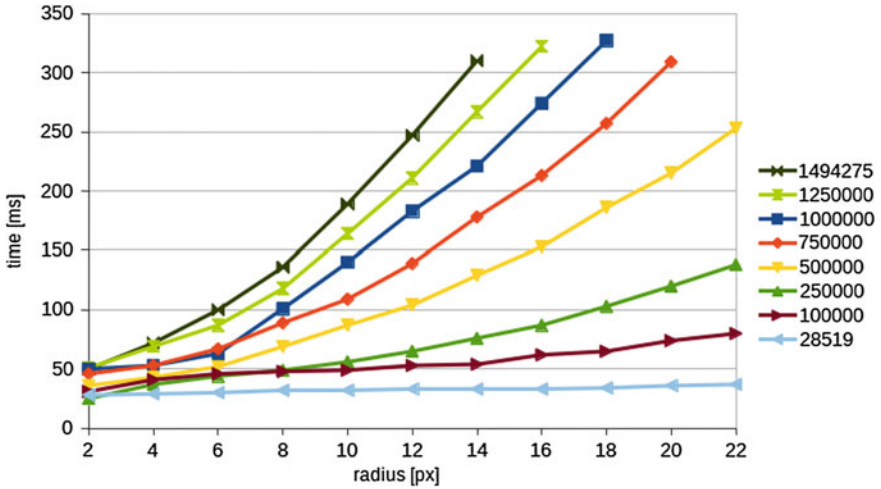


Fig. 4 Rendering times for GPU-based visualization

the usability of the interactive heat map using the GPU-based visualization. We received this comment: “An interactive heat map enables a user to browse large dataset and might reveal hidden patterns. Such insights serve a valuable input to better formulate precise traffic engineering statistical analysis and safety management. State-of-the art tools that we used so far were not able to provide such interactions for the larger datasets”.

Another feedback was gathered from traffic specialists from the Birmingham City Council, fully aware about the traffic accidents hotspots by daily experience. The feedback was positive and especially highlights the interaction responsiveness that increases the usability of the tool and encourages curiosity about the visualized problem and the data. The people with local knowledge usually immediately tend to explore the known places and validate their knowledge about dangerous crossroads and junctions.

6 Conclusions

This paper demonstrates a novel approach of creating kernel density estimation and heat map rendering suitable even for data over 1 million points by using an accelerated graphics hardware—GPU. Benchmark results prove the interactive responsiveness of our system (below 100 ms) even for large data by using a commodity hardware. The solution outperforms the mainstream state-of-the-art systems such as the Leaflet heat map plugin, Google Maps API or ESRI’s ArcGIS Online. Feedback from domain experts proves the benefits and usability of user

interactions that are cannot be achieved by concurrent solutions. The proposed algorithm and workflow was implemented as a part of the WebGLayer javascript open source visualization library.

Acknowledgments The authors of this paper are supported by the European Union's Competitiveness and Innovation Framework Programme under Grant Agreement No. 620533, the OpenTransportNet project.

References

- Anderson TK (2009) Kernel density estimation and k-means clustering to profile road accident hotspots. *Accid Anal Prev* 41(3):359–364. doi:[10.1016/j.aap.2008.12.014](https://doi.org/10.1016/j.aap.2008.12.014)
- Card SK, Newell A, Moran TP (1983) *The psychology of human-computer interaction*. L. Erlbaum Associates Inc., Hillsdale. ISBN:0898592437
- Chainey S, Tompson L, Uhlig S (2008) The utility of hotspot mapping for predicting spatial patterns of crime. *Secur J* 21(1):4–28. doi:[10.1057/palgrave.sj.8350066](https://doi.org/10.1057/palgrave.sj.8350066)
- Dillingham I, Mills B, Dykes J (2011) Exploring road incident data with heat maps. In: *Geographic information science research UK 19th annual conference (GISRUK 2011)*, 27–29 April 2011, University of Portsmouth, Portsmouth
- Dykes J (1998) Cartographic visualization. *J R Stat Soc Ser D* 47(3):485–497. doi:[10.1111/1467-9884.00149](https://doi.org/10.1111/1467-9884.00149)
- Fekete J-D, Plaisant C (2002) Interactive information visualization of a million items. In: *Information Visualization, 2002. INFOVIS 2002. IEEE Symposium on*, pp 117–124. IEEE. doi:[10.1109/INFVIS.2002.1173156](https://doi.org/10.1109/INFVIS.2002.1173156)
- Fotheringham AS, Brunson C, Charlton M (2000) *Quantitative geography: perspectives on spatial data analysis*. Sage, Thousand Oaks
- Fotheringham S, Rogerson P (1994) *Spatial analysis and GIS*. CRC Press, Boca Raton. ISBN:780748401048
- Hardisty F, Robinson AC (2011) The geoviz toolkit: using component-oriented coordination methods for geographic visualization and analysis. *Int J Geogr Inf Sci* 25(2):191–210. doi:[10.1080/13658810903214203](https://doi.org/10.1080/13658810903214203)
- Hennebohl K, Appel M, Pebesma E (2011) Spatial interpolation in massively parallel computing environments. In: *Proceedings of the 14th AGILE international conference on geographic information science (AGILE 2011)*
- Kuo R, An Y, Wang H, Chung W (2006) Integration of self-organizing feature maps neural network and genetic k-means algorithm for market segmentation. *Expert Syst Appl* 30(2):313–324. doi:[10.1016/j.eswa.2005.07.036](https://doi.org/10.1016/j.eswa.2005.07.036)
- Lins L, Klosowski JT, Scheidegger C (2013) Nanocubes for real-time exploration of spatiotemporal datasets. *Visual Comput Graph IEEE Trans* 19(12):2456–2465. doi:[10.1109/TVCG.2013.179](https://doi.org/10.1109/TVCG.2013.179)
- Liu Z, Jiang B, Heer J (2013) Immens: real-time visual querying of big data. In: *Computer graphics forum*, vol 32. Wiley, Hoboken, pp 421–430
- Silverman BW (1986) *Density estimation for statistics and data analysis*. Chapman and Hall, Boca Raton. ISBN:978-0412246203
- Skupin A (2004) The world of geography: visualizing a knowledge domain with cartographic means. *Proc Natl Acad Sci* 101(Suppl. 1):5274–5278. doi:[10.1073/pnas.0307654100](https://doi.org/10.1073/pnas.0307654100)
- Slocum TA (2008) *Thematic cartography and geovisualization*, 3rd edn. Prentice Hall, Upper Saddle River. ISBN:978-0132298346

- Srinivasan BV, Qi H, Duraiswami R (2010) Gpuml: graphical processors for speeding up kernel machines. In: Siam conference on data mining. Workshop on high performance analytics-algorithms, implementations, and applications
- O’Sullivan D, Unwin D (2014) Geographic information analysis, 2nd edn. Wiley, Hoboken. ISBN:978-0-470-28857-3
- Wilkinson L, Friendly M (2009) The history of the cluster heat map. *Am Stat* 63(2):179–184
- Wongsuphasawat K, Pack M, Filippova D, Van Daniker M, Olea A (2009) Visual analytics for transportation incident data sets. *Transp Res Record: J Transp Res Board* 2138:135–145

The Rise of Big Spatial Data

Ivan, I.; Singleton, A.; Horák, J.; Inspektor, T. (Eds.)

2017, XXVII, 408 p. 155 illus., 129 illus. in color.,

Hardcover

ISBN: 978-3-319-45122-0