Defence Research and Development Canada
Recherche et développement pour la défense Canada

CAN UNCLASSIFIED

DRDC | RDDC
technologysciencetechnologie

GeoHexViz—Geospatial visualization using hexagonal binning software

Design reference and instruction manual

Tony Abou Zeidan
Canadian Joint Operations Command

Mark Rempel
DRDC – Centre for Operational Research and Analysis

Defence Research and Development Canada

Reference Document
DRDC-RDDC-2021-D183
December 2021

CAN UNCLASSIFIED

Canada

IMPORTANT INFORMATIVE STATEMENTS

Abstract

Geospatial visualization is an important communication method that is often used in military operations research to convey analyses to both analysts and decision makers. When these types of visualizations include a large amount of point-like data, binning—in particular, hexagonal binning—may be used to summarize the data and subsequently produce an effective visualization. However, creating such visualizations may be frustrating for many since it requires in-depth knowledge of both Geographic Information Systems and analytical techniques, not to mention access to software that may require a paid license, training, and perhaps knowledge of a programming language. In this document we describe GeoHexViz which aims to reduce the time, in-depth knowledge, and programming required to produce publication-quality geospatial visualizations that use hexagonal binning. We describe the high-level design of GeoHexViz, its functional specification, and present four examples that demonstrate the capabilities of GeoHexViz in action. For each, we describe the two methods that GeoHexViz provides to do so: first, a command-line script whose input is a JavaScript Object Notation file that contains the visualization's properties; and second, a Python script that imports and invokes functions found in the software's Python modules.

Significance for defence and security

The significance of GeoHexViz for defence and security is that is it reduces obstacles that may hinder an analyst from producing a publication-quality geospatial visualization for inclusion in their scientific documents or presentations. In turn, it has the potential to improve the collective capability of analysts to effectively communicate with decision makers and each other.

CAN UNCLASSIFIED

Résumé

La visualisation géospatiale est une méthode de communication importante, souvent utilisée pour transmettre des analyses aux analystes et aux décideurs participants à la recherche sur les opérations militaires. Lorsque ces types de visualisation englobent une grande quantité de données ponctuelles, le groupement de données par classe — et plus particulièrement le groupement de données par classe hexagonale — peut être utilisé pour synthétiser les données et ainsi générer une visualisation efficace. Cependant, la création de telles visualisations peut être contrariante puisqu'elle suppose une connaissance approfondie des systèmes d'information géographique et des techniques d'analyse, sans oublier l'accès à un logiciel (sous licence ou pas), une formation et la maîtrise d'un langage de programmation. Dans le présent document, nous vous présentons GeoHexViz, une ressource qui permet de réduire le temps, les connaissances et la programmation nécessaires à la production de visualisations géospatiales diffusables basées sur des groupements de données par classe hexagonale. Nous décrivons la conception de haut niveau et la spécification fonctionnelle de GeoHexViz, et présentons quatre exemples qui illustrent les capacités de GeoHexViz. Pour chacun de ces exemples, nous expliquons les deux méthodes proposées par GeoHexViz : 1) un script de commandes dont le fichier d'entrée est un fichier de notation objet JavaScript (JSON) qui contient les propriétés de la visualisation ; 2) un script Python qui importe des fonctions trouvées dans les modules Python du logiciel et les exécute.

Importance pour la défense et la sécurité

GeoHexViz revêt une importance pour la défense et la sécurité, car il réduit les obstacles qui peuvent empêcher un analyste de produire une visualisation géospatiale d'assez bonne qualité pour être publiée, puis incluse à ses documents ou présentations scientifiques. De ce fait, GeoHexViz a le potentiel d'améliorer la capacité collective des analystes à communiquer efficacement entre eux et avec les décideurs.

Table of contents

List of figures

List of tables

Acknowledgements

Thank you to Nicholi Shiell for his input in testing and providing advice for the development of this package and its supporting documents.

1 Introduction

Visualizing geospatial data—data with a location-based attribute [1]—helps analysts to better understand data and communicate their analyses to both colleagues and decision makers [2]. In particular, geospatial visualization helps to "communicate how different variables correlate to geographical locations by layering these variables over maps" [3]. In military operations research, geospatial visualizations have been used in a wide range of analyses. Examples include:

- Feibush et al. [4] discussed software that uses geospatial visualizations to help commanders coordinate thousands of units over a large geographic region;
- Laskey et al. [5] used geospatial visualizations to depict how different terrain impacts the performance of vehicles;
- Kovařík [6] used geospatial visualization to communicate the results of an analysis concerning optimal placement of tactical or non-permanent helicopter sites in varying geographic regions;
- Connable [7] used geospatial visualizations to represent positioning of allied and enemy units on a battlefield;
- Goodrich et al. [8] discussed software that uses geospatial visualizations to automate the process of watershed analysis "to aid installation managers in sustaining their mission requirements in support of testing and training" (p. 1); and
- Hunter et al. [9] used geospatial visualization to convey the results of an analysis concerning the number of survivors of a major maritime disaster as a function of incident location in the Arctic.

Regardless of the application, "[g]eospatial analysis tools ... [help analysts] to visualize data in a way that [aids] a commander's thought process" [7].

Creating geospatial visualizations is often time-consuming and laborious [10]. This is due to that in-depth knowledge of Geographic Information System (GIS) concepts is required to use GIS software, and hence to create geospatial visualizations [11]. For example, an individual must decide which map projection to use, the colour scheme, the basemap, and in some cases how to organize the data in layers. There are many software applications that may be used to create geospatial visualizations, such as ArcGIS, QGIS, and D3 [12, 13, 14]. ArcGIS provides a wide range of capabilities, but requires a paid license and a solid foundation in geospatial information processing [15]. In contrast, QGIS is free and open-source, but also requires an in-depth knowledge of geospatial information processing to be used effectively [16]. An alternative approach, developed in the last decade, is D3. In addition to an understanding of geospatial concepts, it also requires a knowledge of JavaScript [17]. More recently, a new library that has been introduced is the Plotly graphing library. Plotly provides easy-to-use geospatial graphing functions [18]; however, it too requires knowledge of a programming language. Common across these applications is the requirement to have knowledge of geospatial concepts, and acquiring this knowledge has been identified as a significant challenge [19].

In addition to being time consuming to create, geospatial visualizations often require analysts to have specialized knowledge of analytic techniques. One of these techniques is binning in which a grid is placed over a data set and the individual data points are grouped by grid cell [20]. This method is used when it is difficult to visualize geospatial point-like data sets; in particular, when the number of points is large, they become cluttered and cannot be easily distinguished from one another [20, 21]. In order to provide an accurate representation of the data, an analyst must choose a versatile grid type. There are many grid types available, such as circular, rectangular, and hexagonal. A circular grid is optimal for analysis purposes because circles are accurate for sampling, but does not provide a continuous grid [22]. Rectangular grids are simple to implement; however, may not be suitable when investigating connectivity or movement [23]. A hexagonal grid is often selected because its more visually appealing than other grid types [24], and shares many of its properties with a circular grid [22]. In addition, hexagonal grids offer many advantages including: hexagons have the same number of neighbours as they does edges; the center of each hexagon is equidistant from the centers of its neighbours (which helps when analyzing connectivity or movement); and hexagons tile densely on curved surfaces, resulting in lower edge effects (reducing analytic bias) [22]. The previously mentioned GIS systems provide functionality to perform hexagonal binning, albeit access to this functionality is often limited due to the issues described above.

This Reference Document describes the GeoHexViz software package, whose aim it is to reduce the time and in-depth knowledge required to produce publication-quality geospatial visualizations that use hexagonal binning. GeoHexViz, which is built on top of Plotly, allows an analyst to produce a publication-quality visualization in two ways. First, the package allows a user to generate a visualization via running a pre-existing command-line script whose input is a single JavaScript Object Notation (JSON) file that defines the properties of the visualization. Second, a user can generate a visualization by writing a Python script that imports and invokes functions on objects found in the software's Python modules. Both methods require that the user provide only two arguments. The first argument is a reference to the data—which is a file path, or may be a DataFrame [25] or GeoDataFrame [26] when using the second option. The second argument is a reference to the columns within the data that define the latitudes, longitudes, and the value associated with each. If no value column is present, the default of each data entry is set to one.

The remainder of this Document is organized as follows. Section 2 presents the high-level design specification of the GeoHexViz software package. Next, Section 3 discusses how the software works—from data preparation to generating the visualization. Then, Section 4 provides a set of examples that demonstrate how to use the software. Next, Section 5 describes three limitations of GeoHexViz—an issue surrounding anti-meridian crossing geometries, an issue surrounding colour bar positioning, and an issue surrounding the generation of hexagons. Finally, Section 6 brings forth concluding remarks.

2 Design specification

The aim of GeoHexViz is to simplify the production of publication-quality geospatial visualizations that utilize hexagonal binning. To do this, user specifies a set of *layers*—where each layer is defined as a "[group] of point, line, or area (polygon) features representing a particular class or type of real-world entities" [27]—to be visualized. At a minimum, the user must specify one layer, the *hexbin layer*, through two arguments—a reference to the point-like data to be hexagonally binned, and references to the columns containing latitudes, longitudes, and value at each coordinate. Given the hexbin layer, GeoHexViz will produce a hexagonally binned geospatial visualization which is output in PDF format for publications. Additional layers may be specified, such as regions, grids, and outlines, and the output may be generated in PNG, JPEG, WEBP, SVG, or EPS formats.

If the output visualization is not satisfactory, GeoHexViz allows a user to adjust features of the plot. These features include:

- **scale**: the data displayed in the visualization may be on a linear (default) or logarithmic scale;
- **colour scale**: the colour scale of the visualization may be continuous (default) or discrete;
- **focus**: the visualization may have no focal point (default), showing a view of the whole Earth, or may be focused on the data; and
- **filtering**: all of the data may be present in the visualization (default) or may be clipped to a geographic region.

In addition, the user can change other properties that are passed directly to Plotly: border colour, land colour, sea colour, and figure size. The high-level process used by GeoHexViz, including inputs, user assessment, and adjustment of a visualization via a feedback loop, is depicted in Figure 1.

GeoHexViz may be used to create a visualization in two ways. First, the user can use GeoHexViz's command-line script—GeoHexSimple—to input a JSON file that contains properties for the visualization. The purpose of the command-line is to give non-technical users a simple interface to build a hexagonally binned plot. Second, the user can generate a visualization via importing and invoking functions found in the software's Python modules. When using the second method, the reference to the data may be a DataFrame [25] or GeoDataFrame [26] object. If the input reference is a GeoDataFrame, the package does not *need* latitude or longitude columns. Instead, the input to the software will be the entries within the geometry column (it is up to the user to ensure that these are valid geometry types).

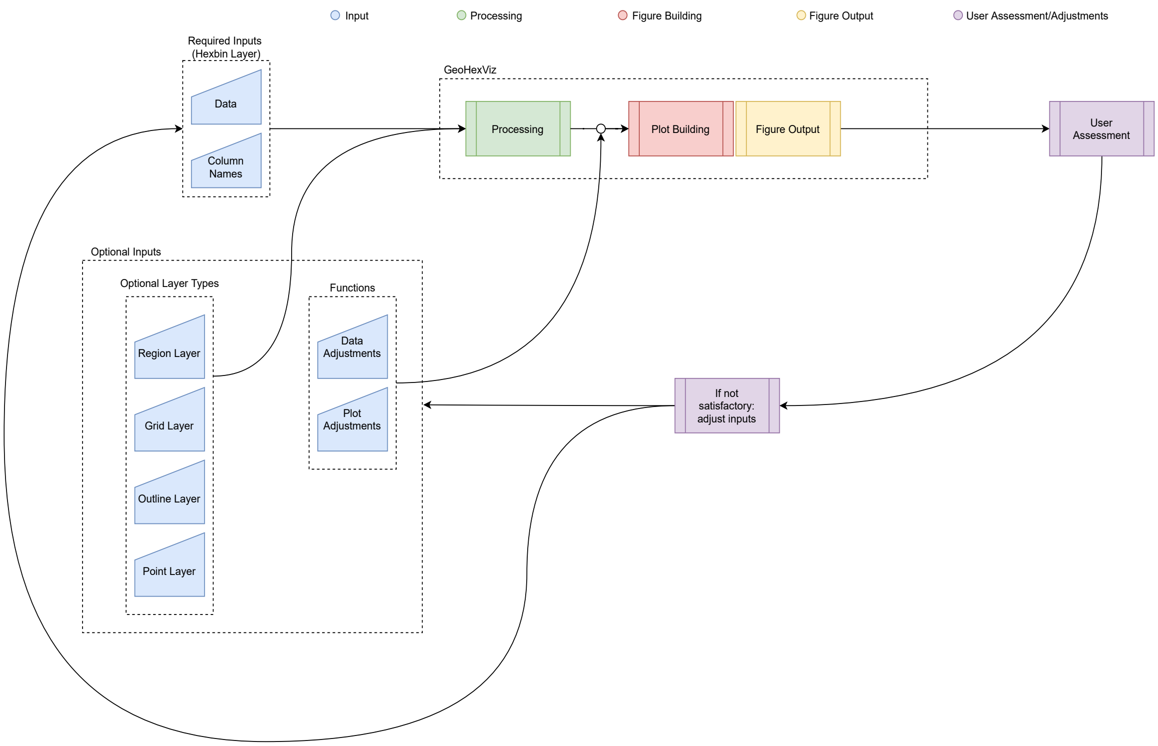


Figure 1: *High-level process used by of GeoHexViz, including inputs, user assessment, and adjustment.*

3 Functional specification

This section describes the input, behaviour, and output of the GeoHexViz. First, the overall design of the software is described in Subsection 3.1. Next, Subsection 3.2 discusses the required and optional input properties for GeoHexViz. Subsequently, Subsection 3.3 presents functions and adjustments that the user may invoke before the plot building step (not applicable when using the GeoHexSimple command-line script) or after the user assessment step seen in Figure 1. Then, Subsection 3.4 discusses the two methods of using GeoHexViz. Afterwards, Subsection 3.5 describes the steps involved in the processing of the input data, and Subsection 3.6 discusses the steps involved in the output of visualizations.

3.1 Software design

The GeoHexViz package consists of four components which are:

1. the functions behind loading the user's input;
2. the functions behind processing the user's input data;
3. the functions behind preparing the visualization; and
4. the functions behind outputting the user's visualization.

Figure 2 depicts these four components. The left section (blue) shows the required inputs for each type of layer. For example, the required inputs for the hexbin layer (labelled A in the figure) are a data reference and column headers. The green section (second from the left) presents the processing steps for each type of layer. For example, the hexbin layer has seven processing steps from the conversion of input to the adding of hexagonal geometries (steps A:1 to A:7). The red section (third from the left) displays the processing steps that take place when the figure is being built and prepared for output. For example, the hexbin layer must be put into GeoJSON form and then made into a Graph Trace (steps A:8 and A:A/B). Finally, the yellow section (right) presents the two methods of figure output (step C).

In order to perform these processes, GeoHexViz relies upon existing libraries listed in Table 1. The Uber H3 library [28] is responsible for the generation of the hexagonal grids. The library was selected as it can quickly retrieve the hexagons that fall within a given polygon, or that correspond to a geographical coordinate. GeoPandas is a library that extends Pandas for geospatial information processing [29, 30]. It uses Shapely to define geometries, and their boundaries [31]. GeoHexViz uses these libraries to internally manage and process the user's geospatial data and hexagonal grid. Plotly is an open-source graphing library that supports the visualization of geospatial data [18]. The visualization portion of GeoHexViz is built on top of Plotly which is used to incrementally build a plot and visualize the output. Using Plotly Kaleido, GeoHexViz is able to output the geospatial visualization in a variety of formats such as PDF, EPS, JPEG, etc.

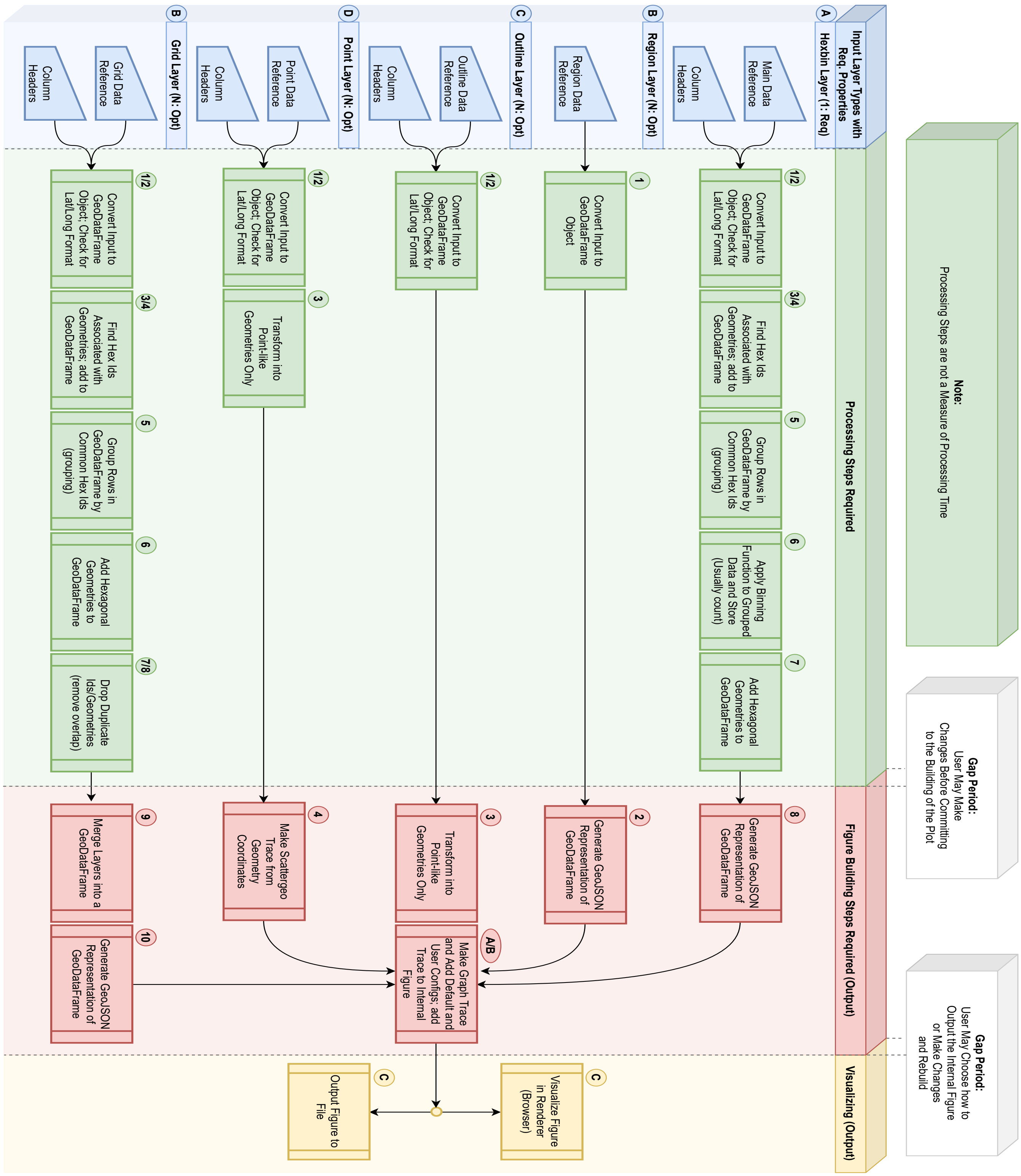


Figure 2: Software flow (Opt: Optional, Req: Required).

Table 1: *Libraries incorporated into the GeoHexViz package with use description (Ver:Version, Bld:Build, Ch:Channel).*

Library			Use Description
Name	Information		
Pandas	**Ver:**	pandas(1.2.2)	Pandas makes the processing of data more efficient by using database style structures and functions [30]. In this application pandas is used to manipulate data; mostly used when performing operations on data sets.
	Bld:	py39h2e25243_0	
	Ch:	conda-forge	
GeoPandas	**Ver:**	geopandas(0.8.2)	GeoPandas leverages the objects defined in Pandas and integrates them with functions to make the processing of geospatial data more efficient [29]. In this application GeoPandas is used to define and store any geospatial data.
	Bld:	pyhd8ed1ab_0	
	Ch:	conda-forge	
Uber H3	**Ver:**	h3-py(3.7.0)	H3 is a hexagonal hierarchical geospatial indexing system that converts conventional lat/lon coordinates into a special 64-bit H3 index [28]. In this application H3 was used in order to construct hexagonal grids over areas of interest and data sets.
	Bld:	pyhd8ed1ab_0	
	Ch:	conda-forge	
Shapely	**Ver:**	shapely(1.7.1)	Shapely is a library for the manipulation and analysis of geometric objects [31]. In this application Shapely is used to facilitate the definitions of regions of interest through Polygon, and Point objects.
	Bld:	py39hadd88af_1	
	Ch:	conda-forge	
Plotly	**Ver:**	plotly(4.14.3)	Plotly is an open-source graphing library [18]. In this application Plotly is used so that results can be visualized and easily integrated into reports and presentations.
	Bld:	pyh44b312d_0	
	Ch:	conda-forge	
	Ver:	kaleido-core(0.2.1)	
	Bld:	h8ffe710_0	
	Ch:	conda-forge	
	Ver:	python-kaleido(0.2.1)	
	Bld:	pyhd8ed1ab_0	
	Ch:	conda-forge	

Table 2: *Properties of the hexbin layer (Req: Required, Qual: Qualitative Data, Quant: Quantitative Data).*

Property	Req.	Description	Default
`data`	Yes	readable data to be hex-binned	-
`latitude_field`	Yes	column containing latitudes	-
`longitude_field`	Yes	column containing longitudes	-
`binning_field`	No	column to bin by	ones column
`binning_fn`	No	function to perform on binned data	quant:sum/count qual:best
`hex_resolution`	No	hex resolution (0-15)	3
`manager`	No	properties to be passed into underlying libraries (see [32])	-

3.2 Input data specification

In order to generate a publication-quality geospatial visualization, GeoHexViz requires a user to specify a set of layers via properties, including both those that are required and optional. At a minimum, the hexbin layer must be specified via its required properties as discussed in Subsubsection 3.2.1. Additionally, optional layers types may be specified as discussed in Subsubsection 3.2.2.

3.2.1 Hexbin layer and its properties

GeoHexViz *requires* the user provide the hexbin layer, which is specified by two properties, to create a visualization. The first is a reference to the data which may be accepted in a comma-separated values (.csv) format, or one of many GIS formats that are accepted by GeoPandas, i.e., Shapefile (.shp), GeoPackage (.gpkg), and GeoJSON (.json). The second is the names of the columns that represent latitude, longitude, and value associated with each. If no value is associated with each location, the program defaults to one for each entry. Further optional properties that define the hexbin layer may be specified. These, along with the required properties, are listed in Table 2.

3.2.2 Optional layers

There are a variety of optional layer types that can be passed into GeoHexViz. In some instances the user may want to highlight a region (or regions) of interest. In these cases the user can input a variable amount of *region-type layers*, which are plotted as filled polygons on the map. These region-type layers are plotted as Plotly Choropleth traces (see [32]). There exists *outline-type layers* which behave similarly to region-type layers; they are plotted as empty polygons via Plotly Scattergeo traces instead (see [33]). Finally, a user may want to display scatter data on top of the hexagonally binned data. In these cases, the user can add *point-type layers* which are plotted as Plotly Scattergeo traces. The required and optional properties for each optional layer type can be seen within Table 3.

Note that GeoHexViz does not require any `latitude_field` or `longitude_field` properties when the data argument contains columns labelled `latitude` and `longitude` or some alias of each respectively. Also, the properties `latitude_field` and `longitude_field` are not required when there is Shapely geometry present within the loaded data. This is the case most of the time when being read from a GIS format.

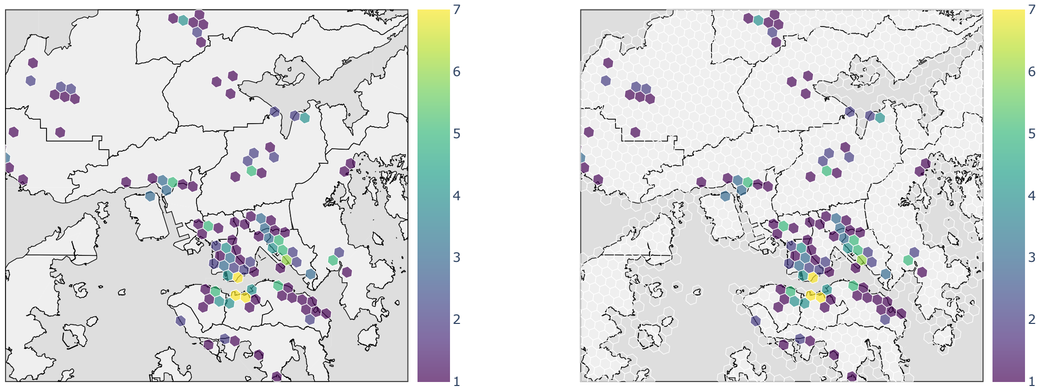
3.2.3 Extended hexagonal tiling (grid layers)

In some instances, the user's data may not cover the entire area of interest and hence it does not form a continuous grid. In such cases, the user can add any amount of *grid-type layers* which act as a way to extend the hexagonal tiling and form a continuous grid which help to show connectivity or movement. All other layer types have a `manager` property that stores the arguments that are passed to Plotly, specific to each individual layer. Extended grid layers do not have individual `manager` properties, as they are merged into a single layer upon the plot being built. Instead, the Plotly arguments are stored in a separate collection called the `grid_manager` which is discussed in Section 4. The required and optional properties for grid-type layers are listed within Table 3.

Figure 3 demonstrates how extended hexagonal tiling can be useful. Both sub figures depict electric vehicle charging stations over the various territories of Hong Kong. In Figure 3a, the data is plotted without a grid-type layer, and given the data does not show a continuous grid. In Figure 3b, the data is plotted again along with a grid-type layer, resulting in continuous grid being formed.

Table 3: *Properties of optional layer types (Req: Required); note the required properties vary for each layer type, the required and optional properties for the hexbin layer are listed in Table 2.*

Type	Property	Req.	Description	Default
Region	`name`	Yes	A name for the layer to be referred to as (continent/country names accepted)	-
	`data`	Yes	Reference to the data that defines this region	-
	`manager`	No	Properties to be passes into underlying libraries (see [32])	-
Outline	`name`	Yes	A name for the layer to be referred to as (continent/country names accepted)	-
	`data`	Yes	Reference to the data that defines this outline	-
	`latitude_field`	Yes	Column in the data containing latitudes	-
	`longitude_field`	Yes	Column in the data containing longitudes	-
	`to_boundary`	No	Converts the geometry in present in the data to one boundary (experimental)	False
	`manager`	No	Properties to be passed into the underlying libraries (see [33])	-
Point	`name`	Yes	A name for the layer to be referred to as	-
	`data`	Yes	Reference to the data that defines this set of points	-
	`latitude_field`	Yes	Column in the data containing latitudes	-
	`longitude_field`	Yes	Column in the data containing longitudes	-
	`text_field`	No	Column in the data containing text for each entry	-
	`manager`	No	Properties to be passed into the underlying libraries (see [33])	-
Grid	`name`	Yes	A name for the layer to be referred to as	-
	`data`	Yes	Reference to the data defining the geometries for this grid	-
	`latitude_field`	Yes	Column in the data containing latitudes	-
	`longitude_field`	Yes	Column in the data containing longitudes	-
	`hex_resolution`	No	Resolution of hexagons (0-15)	3



(a) Without extended hexagonal tiling. *(b) With extended hexagonal tiling.*

Figure 3: *Example GeoHexViz extended hexagonal tiling; density of electric vehicle charging stations in Hong Kong (hexagonal resolution = 8, average hexagon area = 0.74km^2).*

3.3 Optional adjustments

There exists a range of functions the user can perform once they have supplied the layer(s). These functions fall into two categories being data adjustments and plot adjustments discussed in the following subsections. When using GeoHexViz in a Python module, any of these functions may be used at the first Gap Period highlighted in Figure 2. In contrast, using the JSON input mechanism, both Gap Periods are not accessible to the user. As a consequence, the user may use these functions by referring to them in the JSON file. Examples of using the JSON input mechanism are discussed in Section 4 and Python approach are given in Annex A through Annex D.

3.3.1 Plot adjustments

Functions that are considered plot adjustments are ones that only modify the properties that are passed to Plotly. The first of these functions is the `adjust_focus` function which takes a query of layers currently present within GeoHexViz and shifts the focus of the plot to the geometries that were found within the result of the query. If the argument passed into the function is `hexbin+outlines` the function will adjust the focus of the plot to the geometries within the hexbin layer and any outline-type layers. The second of these functions is the `adjust_opacity` function which adjusts the colour scale of the hexbin layer to match the opacity of the colours within the plot (as Plotly does not do this by default). The third plot adjustment function is the `discretize_scale` function which makes the continuous colour scale of the plot into a segmented colour scale.

3.3.2 Data adjustments

Functions that are considered data adjustments are ones that modify the data directly, which in turn may or may not change the resulting plot. The first of these functions is the `remove_empties` function. This function removes empty rows in the hexagonally binned data and adds these empty rows to a grid-type layer. These empty rows are defined by the `empty_symbol` property which is zero by default; when passed, the rows that have a value matching the `empty_symbol` argument are removed. These empty rows are then automatically added to a grid-type layer—a layer containing empty hexagons with no colour mentioned in Subsubsection 3.2.3. The second of these functions is the `logify_scale` function which applies the log function to all of the values in the hexagonally binned layer. This function could also be considered a plot adjustment because it does change some of the Plotly colour bar properties. The third data adjustment function is the `clip_datasets`. This function clips or filters the data to a certain region defined by another optional layer. If the function is passed the arguments `hexbin` and `regions+outlines`, then the hexbin layer is clipped to any region- and outline-type layers. The final data adjustment function is the `simple_clip` function which is a wrapper for the `clip_datasets` function. This function by default clips the hexbin and grid-type layers to region- and outline-type layers.

3.4 Input mechanisms

As mentioned in Section 2, GeoHexViz provides two methods to build a visualization. The first method involves the user running the GeoHexSimple command-line script whose input is a JSON file. The JSON file describes the contents of the visualization in three portions: the input data, functions/adjustments, and output. In the input portion of the JSON file, the user specifies the layers as collections of the properties for the layer types mentioned above. When inputting the optional layer types within the JSON, they are stored in separate collections where the key of each layer is the name to which it is referred to as in the JSON file. The second method involves the user invoking functions provided by GeoHexViz in a Python module of their own. Examples using the JSON input mechanism are found in Section 4, and the corresponding Python code for each example is found in Annex A through Annex D.

Note that when using the command-line script, the order of the objects within the JSON file does not matter. The user can use the functions mentioned within Subsection 3.3 by inputting them into the `functions` object. The user may output or visualize the final figure by adding the `output`, and `display` objects to the end of their JSON file. The user may also alter the internal figure and manage the properties that are to be passed into Plotly by adding the `figure_manager` object to the JSON file. If the function that the user intends to invoke takes one or no *required* members, they can specify it in short form. For example, if the user does not wish to crop the output visualization, the output path can be specified as:

```
"output": "<output path>" % output has one required member
```

If the user wishes to adjust the focus of the plot (focusing on the hexbin layer alone), they can specify it like:

```
"functions": {
  "adjust_focus": true % adjust focus has no required members
}
```

Finally, if the user wishes to display the visualization interactively in the web browser, they can specify it in short form:

```
"display": true % display has no required members
```

More complex plots may require looking into the documentation for this project and the documentation of its underlying libraries.

3.5 Processing

This section focuses on the processing steps corresponding to the hexbin layer. The processing steps for other layers are depicted in Figure 2. Once the input data and properties are given, the data is loaded, processed, and hexagonally binned by the software. The processing steps for the hexbin layer found in Figure 2 have been expanded into higher detail within Figure 4.

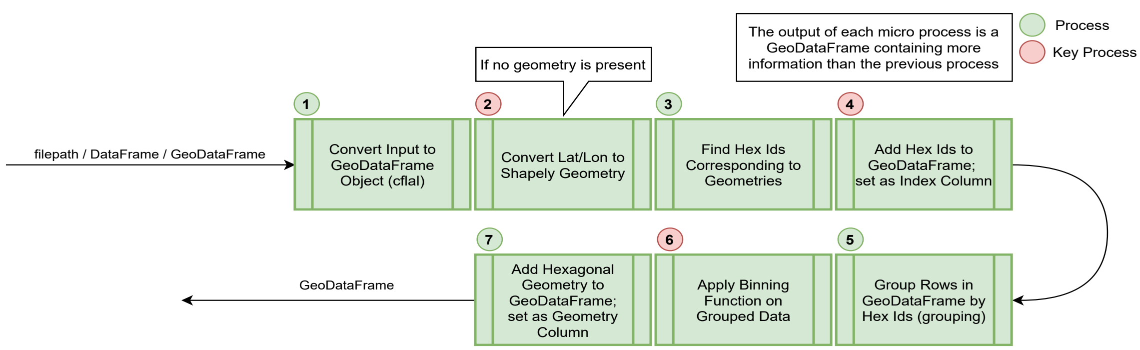


Figure 4: *Processing of the hexbin layer.*

The first step involves loading the data into a usable form. The input data can be in the form of a file path, DataFrame, or GeoDataFrame. In order to maintain consistency, the software converts all of these formats (if applicable) into GeoDataFrame objects. In step two, the provided latitude and longitude columns are used in order to create the entries within the geometry column of the GeoDataFrame (if the column isn't full already). This is a key process because at the end of its execution, the data is stored internally in a form that GeoHexViz considers valid; it ensures that geospatial operations can be performed on the data. Next, in step three the Uber H3 library is used to retrieve a set of ids from the geometry in the data that correspond to hexagons on the globe. Then, in step four these hexagonal ids are added to the index column of the GeoDataFrame. At the end of step four's execution, the hexagonal ids are stored within the data; each data entry is indexed by an id. At this point, the data has had a hexagonal grid placed over it. In step five, the data is grouped by common hexagonal id. Next, in step six this grouped data has the binning function applied to it, and the result is stored in a new column. At this point, the data has been hexagonally binned. The processing is completed in step seven where the geometry for each hexagon is retrieved from the Uber H3 library and is stored within the GeoDataFrame. In the last step, the geometric definitions of each hexagonal id within the DataFrame are retrieved. These geometries are stored in a new column within the DataFrame.

3.6 Output

After the layers have been processed, GeoHexViz generates the visualization. GeoHexViz first builds the plot based on the current state of the internal data. The figure building steps that are taken for each layer type are depicted in Figure 2. For all layer types, the figure building process includes three key steps. First, the data is turned into a form that is usable by Plotly graph objects. This form is either a GeoJSON (for Choropleth traces) or a list of latitudes and longitudes (for Scattergeo) traces. Second, these inputs are passed into Plotly objects; default and user configurations are added to the objects. Finally, these Plotly objects are added to the internal figure. These plot building processes are internal; when invoking functions from a Python module, the user needs to call `finalize()` when they decide to commit to the current state of the data within GeoHexViz.

At this point the figure building process is complete. The user now has one of two options. With the input of a path, the user can output the figure in file form. This process uses Plotly Kaleido which supports various formats such as PNG, JPEG, EPS, and PDF. The second option is that the user can visualize the figure in a renderer of their choosing (by default this is set to the browser). Using this method the user can interact with the plot (zoom, drag, see data within each hexagon).

4 Examples

This section provides four examples of GeoHexViz being used to generate visualizations of hexagonally binned data, where the data is either qualitative or quantitative. Each visualization in this section was created using the `GeoHexSimple` command-line script, as described in Subsection 3.4. This section assumes the reader has sufficient knowledge of the JSON file format; if not, see [34]. In this section, references to JSON elements (objects and members) are given in `this font`; clicking on these elements will direct the reader to the line within the code that it is defined. Each example is structured as follows. First, the data set used for the visualization is described. Next, the JSON file used to generate the visualization is described in segments. Finally, these segments are then pieced together into a complete JSON file. The corresponding Python code to generate each visualization is found in Annex A through Annex D.

4.1 Search and Rescue

The first example concerns a data set containing the locations of search and rescue (SAR) incidents over the Canadian landmass. The data was taken from the Search and Rescue Mission Management System [35], and contains 131 867 incidents between 2008 and 2019. The visualization generated by GeoHexViz is given in Figure 5.

To make this visualization the required properties for the data to be hexagonally binned must be specified. This is done by adding the `hexbin_layer` object to the JSON file.

```
{
  "hexbin_layer": {
    "data": "<data file location>",
    "hex_resolution":4,
    "manager": {
      "colorscale": "Viridis",
      "colorbar": {
        "x": 0.8325
      }
    }
  },
```

In this example, the `hexbin_layer` object has three members: `data`, `hex_resolution`, and `manager`. The `data` member is a full path to the location of the data. The `hex_resolution` member specifies the size of the hexagons to be used within the plot. This number can range from 0 to 15 and is defined in [28], where 0 represents the largest hexagon size, and 15 represents the smallest hexagon size. Finally, the `manager` member, which itself is an object, specifies properties that are passed to Plotly. In this case there are two being `colorscale`, and `colorbar`. The `colorscale` member specifies the colour scale to be used within the plot. By default, Plotly only allows the named colour scales to be continuous. GeoHexViz overrides this behavior and allows all named colour scales available from Plotly; for the full list of input options see [36, 37, 38]. The `colorbar` member is a collection of items that control different properties of the colour bar, such as background colour, border colour,

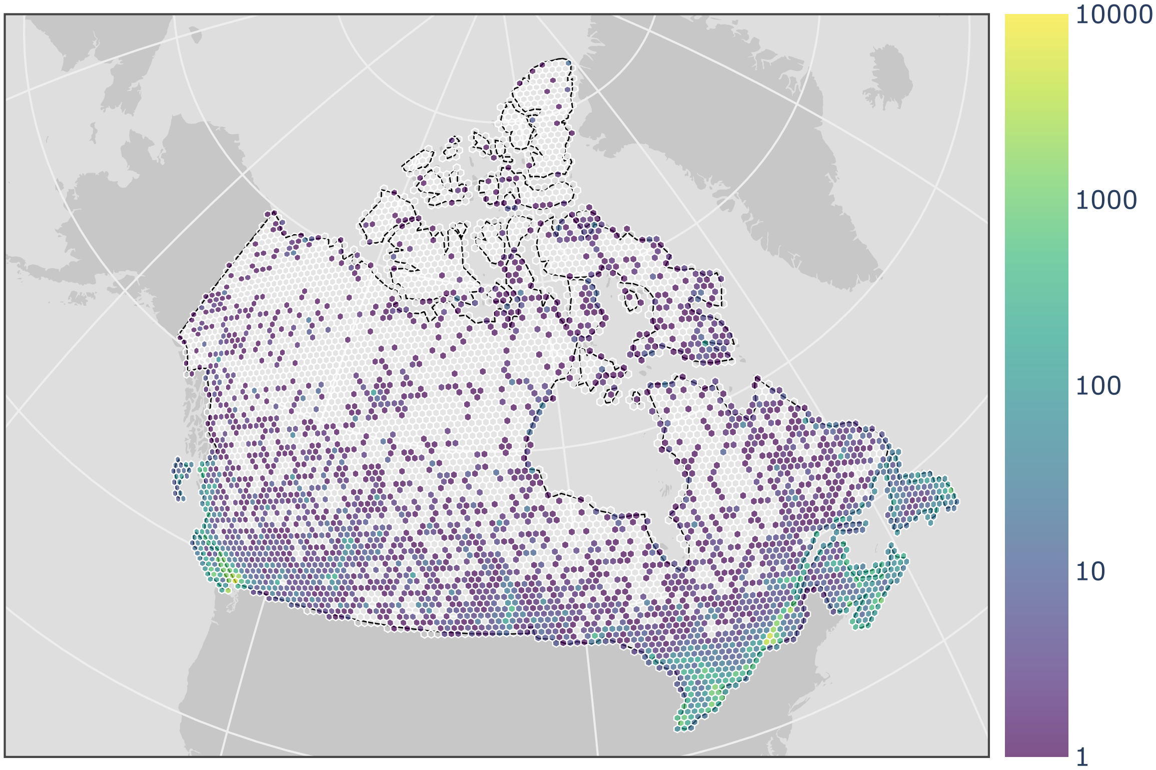


Figure 5: *Density of SAR incidents (Canada: 2008 to 2019—hexagonal resolution = 4, average hexagon area equals 11770 km^2).*

and thickness. In this example, the `x` value of the colour bar is being set which specifies the positioning of the colour bar; as the value goes from 0 to 1, the colour bar moves from left to right. For the full list of colour bar properties that can be passed, see [39].

Next, since the region of Canada is to be highlighted, the `regions` object is added to the JSON file.

```
"regions": {
  "sample_Region_CANADA": {
    "data": "CANADA"
  }
},
```

In this object there can be many defined regions, but for the sake of this visualization only one is needed. This region is defined under the object `sample_Region_CANADA`, where the reference to the `data` defining the region is `CANADA`. GeoHexViz recognizes the name of a country or continent as given by [40] and automatically retrieves the geometries defining it. Note that `sample_Region_CANADA` is the name that the layer will be referred to as, and could be something else.

Next, an extended grid layer is added to form a continuous grid. This is done by adding the `grids` object to the JSON file.

```
"grids": {
  "sample_Grid_CANADA": {
    "data": "CANADA",
    "convex_simplify": true
  }
},
```

Similar to the `regions` object, a grid referred to as `sample_Grid_CANADA` is specified, where its `data` member is also `CANADA`. Due to that the H3 package supplies hexagons whose centroids are within the polygons, the polygon passed may not be completely filled with hexagons. When set to true, the `convex_simplify` property attempts to fix this by expanding the polygon that was passed and then generating the grid.

Next, a set of functions are specified within the JSON file under the `functions` object. See Subsection 3.3 for the full list of functions that can be performed.

```
"functions": {
  "clip_layers": {
    "clip": "hexbin+grids",
    "to": "regions"
  },
  "adjust_focus": {
    "on": "regions",
    "buffer_lat": [0, 3]
  },
  "logify_scale": {
    "exp_type": "r"
  }
},
```

The first object `clip_layers` specifies the data is to be clipped only to the region of Canada. Specifically it does this through the members in the object; the `clip` member specifies what layers to clip and the `to` member specifies the layers to act as the boundary of the clip. In this case, the `clip` member is `hexbin+grids` which refers to the hexbin layer and any grid layers present. The `to` member is `regions` which refers to any region layers present. The second object `adjust_focus` specifies that the plot be focused on the region of Canada (slightly shifted). The `on` member specifies which layers to focus on; in this case it is specified to `regions` which refers to any region layers present. The `buffer_lat` member specifies two numbers that will be added to the lower and upper values of the automatically calculated boundary, i.e., if the automatically calculated latitude range was from 0 to 50, with a `buffer_lat` member of [10, 20], the resulting latitude range would be from 10 to 70. The final object `logify_scale` specifies that the plot use a logarithmic scale (using raw text). The `exp_type` member specifies what type of exponent is to be used in the colour bar; the value `r` means that the raw numbers will be displayed on the colour bar, i.e., 1, 10, 100, 1000, etc. The possible properties for each function are described in the official documentation.

Finally, the output location of the visualization is specified in the JSON file through the `output` object.

```
  "output": {
    "filepath": "<output path>",
    "crop_output": true
  }
}
```

The first member, `filepath`, specifies the destination of output visualization; the extension of the file path determines the file type. The second member `crop_output` specifies that the output visualization be cropped via PdfCropMargins [41]. When set to true, the `crop_output` member requires that the user have PdfCropMargins, alongside its dependencies installed in their environment.

The complete JSON file is given below. The Python module translation of this JSON is listed in Annex A.

```
{
  "hexbin_layer": {
    "data": "<data file location>",
    "hex_resolution":4,
    "manager": {
      "colorscale": "Viridis",
      "colorbar": {
        "x": 0.8325
      }
    }
  },
  "regions": {
    "sample_Region_CANADA": {
      "data": "CANADA"
    }
  },
  "grids": {
    "sample_Grid_CANADA": {
      "data": "CANADA",
      "convex_simplify": true
    }
  },
  "functions": {
    "clip_layers": {
      "clip": "hexbin+grids",
      "to": "regions"
    },
    "adjust_focus": {
      "on": "regions",
      "buffer_lat": [0, 3]
    },
    "logify_scale": {
      "exp_type"!: "r"
    }
  },
  "output": {
    "filepath": "<output path>",
    "crop_output": true
  }
}
```

4.2 Mass shootings in the United States of America

In the following example, a data set containing the locations of mass shootings in the United States of America is passed into GeoHexViz.[1] Each row in the data set contains the number of people killed and injured in the incident. The data was taken from the Gun Violence Archive [42] and contains 2001 incidents from July 30th 2017 to September 14th 2021. The visualization of the total number of people killed and injured is given in Figure 6.

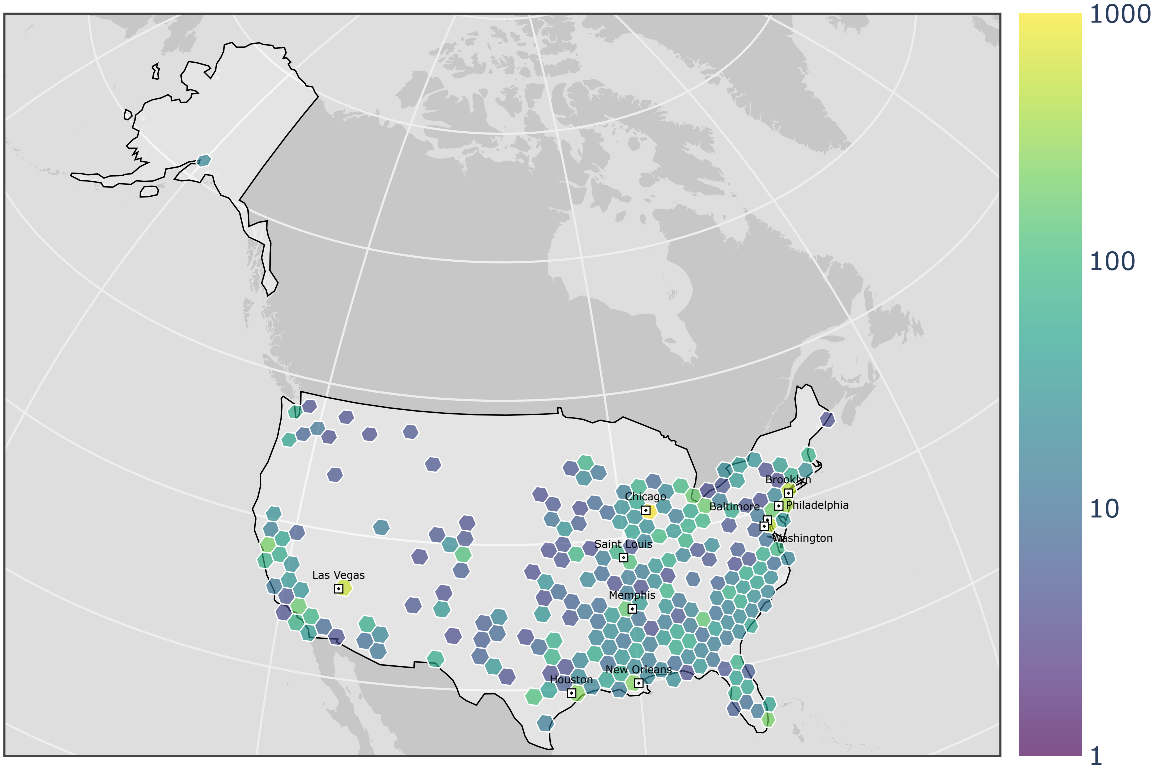


Figure 6: *People killed and injured during mass shootings (United States of America: 30 July 2017 to 14 Sept 2021—hexagonal resolution = 3, average hexagon area = 12392 km^2).*

The steps to building the JSON file for this visualization are very similar to the steps for the previous visualization. The first step is the same; the data that is to be hexagonally binned, alongside its configurations are passed into the `hexbin_layer` object in the JSON file.

```
{
  "hexbin_layer": {
    "data": "<data file location>",
    "hex_resolution":3,
```

[1] A mass shooting is defined as a shooting in which four or more individuals were shot or killed, not including the shooter [42].

```
    "hexbin_info": {
      "binning_field": "killed_injured",
      "binning_fn": "sum"
    },
    "manager": {
      "colorbar": {
        "x": 0.8365
      }
    }
  },
```

For this example, the hexbin_layer object has four members: data, hex_resolution, hexbin_info, and manager. The data member is a full path to the data set containing the mass shooting locations. The hex_resolution member controls the size of the hexagons and is now set to 3. Unlike Subsection 4.1, in this example the data is binned by the incident location and the value displayed is the sum of killed and injured in each hexagon. To do this, the hexbin_info member is added. It does this through its members binning_field, and binning_fn. The binning_field member determines the grouped column to obtain the display value from, and the binning_fn member specifies how this display value is calculated. In this example the binning_field is set to killed_injured which is a column in the data set containing the sum of killed and injured at each incident location.

For this example, the United States of America is to be highlighted as the region of interest. To do this the regions object is added to the JSON file.

```
  "regions": {
    "sample_Region_USA": {
      "data": "UNITED STATES OF AMERICA"
    }
  },
```

A single region is specified within the object and is referred to as sample_Region_USA. The data member of sample_Region_USA is set to UNITED STATES OF AMERICA.

For this example, the epicenters of these incidents are to be displayed over the hexagonally binned data. To do this the points object is added to the JSON file.

```
  "points": {
    "sample_Point_EPICENTERS": {
      "data": "<epicenters file location>",
      "text_field": "city",
      "manager": {
        "textposition": [
          "top center",
          "top center",
          "middle right",
          "top center",
          "top left",
          "bottom right",
          "top center",
          "top center",
          "top center",
          "top center"
        ],
```

```
            "marker": {
              "symbol": "square-dot",
              "size": 4,
              "line": {
                "width": 0.5
              }
            }
          }
        }
      },
```

A single point layer is specified within the object and is referred to as sample_Point_EPICENTERS. The data member of the sample_Point_EPICENTERS is set to a file containing the coordinates and names of the epicenters. The text_field member of the sample_Point_EPICENTERS object is set to the name of the column containing the name of the epicenters. This member controls the text to be displayed on top of each data entry on the map. The manager member of sample_Point_EPICENTERS is an object that contains arguments that are passed to Plotly for this layer. In this example the manager contains two members: textposition, and marker; for the full list of options see [33]. The textposition property controls the positioning of the text to be displayed alongside the scatter data. In this case, since multiple epicenters are near each other, the positioning is set for each epicenter manually; for the full list of options see [43]. The marker member, which itself is an object, controls drawing properties for its associated layer; for the full list of options see [44]. In this example, the marker object is used to change the symbol used, the size of, and the outline width of each data point. To change the symbol for each data point, the symbol member is added to the marker object, and set to square-dot; for the full list of options see [45, 46]. To change the size of each data point, the size member is added to the marker object, and set to 4; for the full list of options see [47]. Finally, to change the outline width for each data point, the line member, which is itself an object, is added marker object. The line object controls various properties for the outline of each data point; for the full list of options see [48]. In this example, the width of the outline is set to 0.5 via the width member of the line object.

Next, a set of functions are specified within the JSON file under the functions object.

```
      "functions": {
        "remove_empties": true,
        "adjust_focus": {
          "on": "hexbin",
          "buffer_lat": [0,15],
          "rot_buffer_lon": -8
        },
        "logify_scale": {
          "exp_type": "r"
        }
      },
```

The first member, remove_empties specifies that empty hexagons be removed from the data. It is set to true as the function has no required arguments. The second member, adjust_focus is an object specifying that the plot be focused on the data. In this case, the `on` member of `adjust_focus` specifies that the plot be focused on the hexbin layer. The `buffer_lat` member of `adjust_focus` specifies that the upper bound of the automatically calculated latitude range be shifted by 15 degrees. The `rot_buffer_lon` member of `adjust_focus` specifies a number to add to the automatically calculated longitude rotation value. For example, if the calculated rotation had a longitude of 8, and the `rot_buffer_lon` value was 2, then the final rotation longitude would be 10. The final member, `logify_scale` is an object specifying that the plot use a logarithmic scale. Once again, the `exp_type` member of `logify_scale` specifies that there be no exponent in the colour bar.

Finally, the output location of the visualization is specified in the JSON file through the `output` object.

```
  "output": {
    "filepath": "<output path>",
    "crop_output": true
  }
}
```

The full JSON structure is as follows. The Python module translation of this JSON is listed in Annex B.

```
{
  "hexbin_layer": {
    "data": "<data file location>",
    "hex_resolution":3,
    "hexbin_info": {
      "binning_field": "killed_injured",
      "binning_fn": "sum"
    },
    "manager": {
      "colorbar": {
        "x": 0.8365
      }
    }
  },
  "regions": {
    "sample_Region_USA": {
      "data": "UNITED STATES OF AMERICA"
    }
  },
  "points": {
    "sample_Point_EPICENTERS": {
      "data": "<epicenters file location>",
      "text_field": "city",
      "manager": {
        "textposition": [
          "top center",
          "top center",
          "middle right",
          "top center",
          "top left",
          "bottom right",
          "top center",
          "top center",
          "top center",
          "top center"
        ],
```

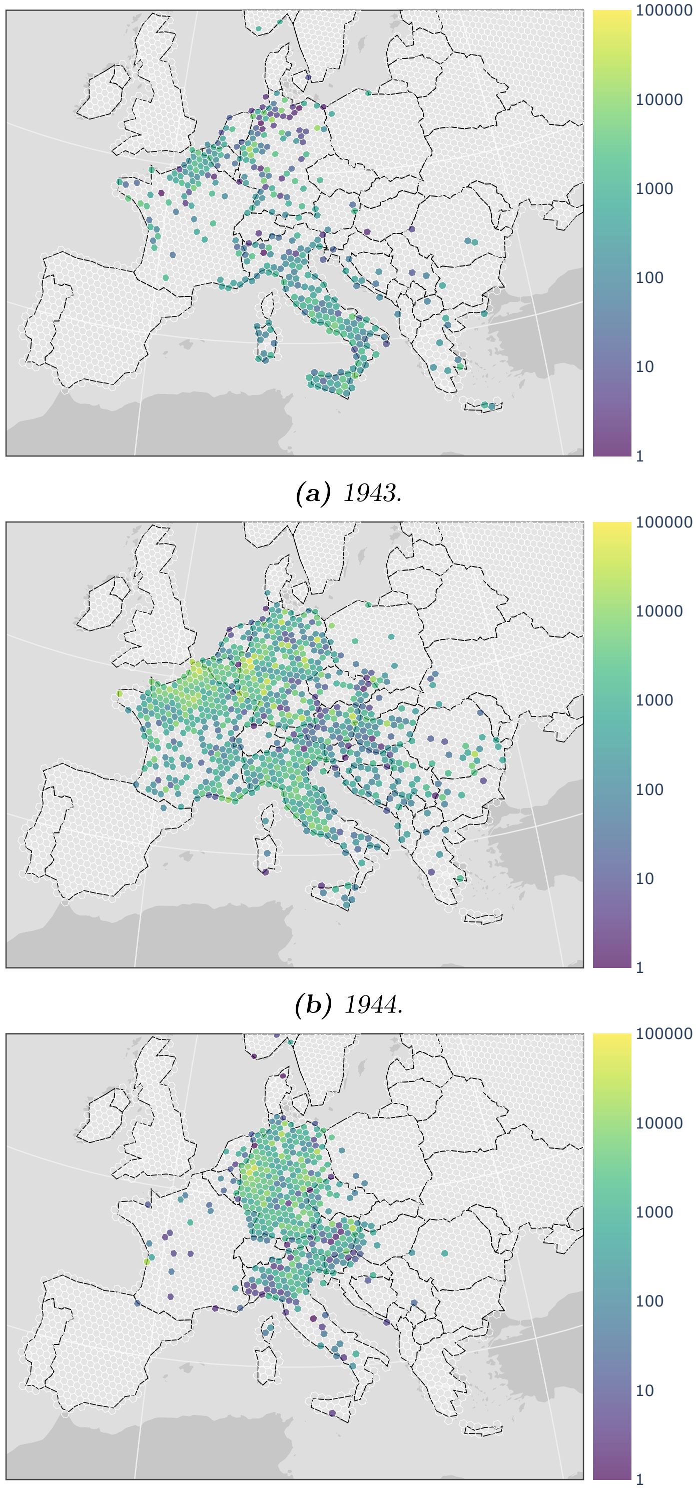
```
        "marker": {
          "symbol": "square-dot",
          "size": 4,
          "line": {
            "width": 0.5
          }
        }
      }
    }
  },
  "functions": {
    "remove_empties": true,
    "adjust_focus": {
      "on": "hexbin",
      "buffer_lat": [0,15],
      "rot_buffer_lon": -8
    },
    "logify_scale": {
      "exp_type": "r"
    }
  },
  "output": {
    "filepath": "<output path>",
    "crop_output": true
  }
}
```

4.3 World War 2 bombings

The next example concerns a data set containing the locations of World War 2 bombings. The data was taken from [49] and split into the years 1943, 1944, and 1945—collectively these years contain 155 175 bombing events in the data. The visualization of the total mass of bombs dropped in each of these years is given in Figure 7.

To make this visualization the required layer and its properties are passed via the `hexbin_layer` object.

```
{
  "hexbin_layer": {
    "data": "<data file location>",
    "hexbin_info": {
      "binning_field": "high_explosives_weight_tons",
      "binning_fn": "sum"
    },
    "hex_resolution":4,
    "manager": {
      "marker": {
        "line": {
          "width": 0.45
        }
      },
      "colorscale": "Viridis",
      "colorbar": {
        "x": 0.82
      }
    }
  },
```



(a) 1943.

(b) 1944.

(c) 1945.

Figure 7: *World War 2 bombings—European Theatre (1943–1945): Total mass of bombs dropped in tons (1943–1945)—mass in tons, hexagonal resolution = 4, average hexagon area = 11770* km^2.

Similar to the previous visualization, the hexbin_layer object has four members. The data member is now a full path to the data set containing the bombing locations. Next, the hexbin_info member is an object specifying how the data is to be binned through its members. The binning_field member of hexbin_info specifies that the data be grouped by high_explosives_weight_tons—a column containing the weight of bombs dropped at each incident location. Once again, the binning_fn member of hexbin_info specifies that the grouped data be summed to retrieve the display value. Next, the manager specifies three members that are passed into Plotly being marker, colorscale, and colorbar. The marker member controls drawing properties for its associated layer (hexbin layer in this case) such as opacity, and other line properties; for the full list of input options, see [50]. In this example the marker member is an object specifying that the line width be set. It does this through the line member which is also an object controlling the properties of line colour and width. In this example, the property width is being set which controls line width.

Since the region of focus in this example is Europe, a region layer containing the European landmass is added to the plot via the regions object.

```
"regions": {
  "sample_Region_EUROPE": {
    "data": "EUROPE"
  }
},
```

The sample_Region_EUROPE is the object defining this region layer. The data member of the region layer is set to EUROPE.

Now extended grid layers are added to fill the gaps within the data and form a continuous grid. Once again, this is done by adding the grids object to the JSON file.

```
"grids": {
  "sample_Grid_EUROPE": {
    "data": "EUROPE",
    "convex_simplify": true
  },
  "sample_Grid_RUSSIA": {
    "data": "RUSSIA",
    "convex_simplify": true
  }
},
```

Since the data spans the European region, we declare a grid layer that also spans this region. This grid layer is defined through the object sample_Grid_EUROPE, whose data member is set to EUROPE. It becomes evident that if the grid layer sample_Grid_RUSSIA is not present, then there are few hexagons present near Russia.

Next, since the line thickness for the hexbin layer has been altered, the line thickness for all grid layers must be the same. This change is made by adding the grid_manager.

```
"grid_manager": {
  "marker": {
    "line": {
      "width": 0.45
    }
  }
},
```

The properties set for this manager's line thickness are identical to those set in the `manager` of the hexbin layer.

Next, since using the `adjust_focus` function does not provide the necessary focus for this plot easily, it is set manually. To do this, the geo layout properties (Plotly) needs to be set; this is done via adding the `figure_geos`. For the full list of properties that can be set for the figure's geo layout, see [51].

```
"figure_geos": {
  "lataxis": {
    "range": [35, 58]
  },
  "lonaxis": {
    "range": [0, 43]
  },
  "projection": {
    "rotation": {
      "lat": 46.63321662159487,
      "lon": 11.21560455920799
    }
  }
},
```

The default projection type of GeoHexViz is the orthographic projection supplied by Plotly. In order to obtain the correct focus for this type of projection there are three properties that need to be set. These properties are the latitude axis range, longitude axis range, and projection rotation. The latitude axis range and longitude axis range specify the range of latitudes and longitudes that appear in the figure once generated. The projection rotation makes the globe rotate to the specified coordinates. First, to set the latitude axis range, the `lataxis` member is added to the `figure_geos` object. The `lataxis` controls many properties for the latitude axis displayed on the figure, such as grid width and grid colour; for the full list of input options, see [36]. For this example, the `range` property of the `lataxis` is set to the range to be displayed in the figure, which is `[35, 58]` or from 35 degrees to 58 degrees. Similarly, to set the longitude axis range, the `lonaxis` member is added to the `figure_geos` object. The `lonaxis` member controls many properties for the longitude axis displayed on the figure; for the full list of input options, see [52]. For this example, the `range` member property of the `lataxis` is set to the range to be displayed in the plot, which is `[0, 43]` or from 0 degrees to 43 degrees. Finally, the projection rotation is set via adding the `projection` member to the `figure_geos` object. The `projection` member controls many properties for the projection that the data be displayed on, such as the type of projection used, the tilt of the projection, and the scale of the projection. For the full list of input options, see [53]. For this example the `rotation` property of the `projection` has its `lat`, and `lon` properties set to the center coordinate of the focus. The `lat`, and `lon` properties get set to `46.63` and `11.22` degrees respectively.

Next, a set of functions are specified by adding the `functions` object to the JSON file. These functions include `clip_layers`, `logify_scale`, and `adjust_focus`.

```
  "functions": {
    "clip_layers": {
      "clip": "hexbin+grids",
      "to": "regions"
    },
    "logify_scale": {
      "exp_type": "r"
    },
    "adjust_focus": false
  },
```

The `clip_layers` function is represented by an object containing the arguments to the function. As the previous examples have done, the `clip` and `to` arguments specify that the hexbin layer and grid layers be clipped to region layers. Once again, the `logify_scale` function is represented by an object whose only member is the `exp_type` argument. This specifies that the plot use a logarithmic scale with no exponents in the colour bar. Next, since the focus has already been specified manually, and the function `adjust_focus` is performed by default, the function needs to be disabled. To do this, the `adjust_focus` member of the `functions` object is set to `false`.

Finally, the output location of the visualization is specified in the JSON file through the `output` object. The `output` object has two members `filepath`, and `crop_output` (set to true) which specify where the visualization is to be output, and that the output be cropped.

```
  "output": {
    "filepath": "<output path>",
    "crop_output": true
  }
}
```

The full JSON structure is given below. The Python module translation of this JSON is given in Annex C.

```
{
  "hexbin_layer": {
    "data": "<data file location>",
    "hexbin_info": {
      "binning_field": "high_explosives_weight_tons",
      "binning_fn": "sum"
    },
    "hex_resolution":4,
    "manager": {
      "marker": {
        "line": {"width": 0.45}
      },
      "colorscale": "Viridis",
```

```
      "colorbar": {
        "x": 0.82
      }
    }
  },
  "regions": {
    "sample_Region_EUROPE": {
      "data": "EUROPE"
    }
  },
  "grids": {
    "sample_Grid_EUROPE": {
      "data": "EUROPE",
      "convex_simplify": true
    },
    "sample_Grid_RUSSIA": {
      "data": "RUSSIA"
    }
  },
  "grid_manager": {
    "marker": {
      "line": {"width": 0.45}
    }
  },
  "figure_geos": {
    "lataxis": {
      "range": [35, 58]
    },
    "lonaxis": {
      "range": [0, 43]
    },
    "projection": {
      "rotation": {
        "lat": 46.63321662159487,
        "lon": 11.21560455920799
      }
    }
  },
  "functions": {
    "clip_layers": {
      "clip": "hexbin+grids",
      "to": "regions"
    },
    "logify_scale": {
      "exp_type": "r"
    },
    "adjust_focus": false
  },
  "output": {
    "filepath": "<output path>",
    "crop_output": true
  }
}
```

4.4 Forest fires

The following example concerns the locations of forest fires in the United States of America. The data was taken from [54], and contains 1291 incidents during the year of 2017. The visualization generated by GeoHexViz is given in Figure 8.

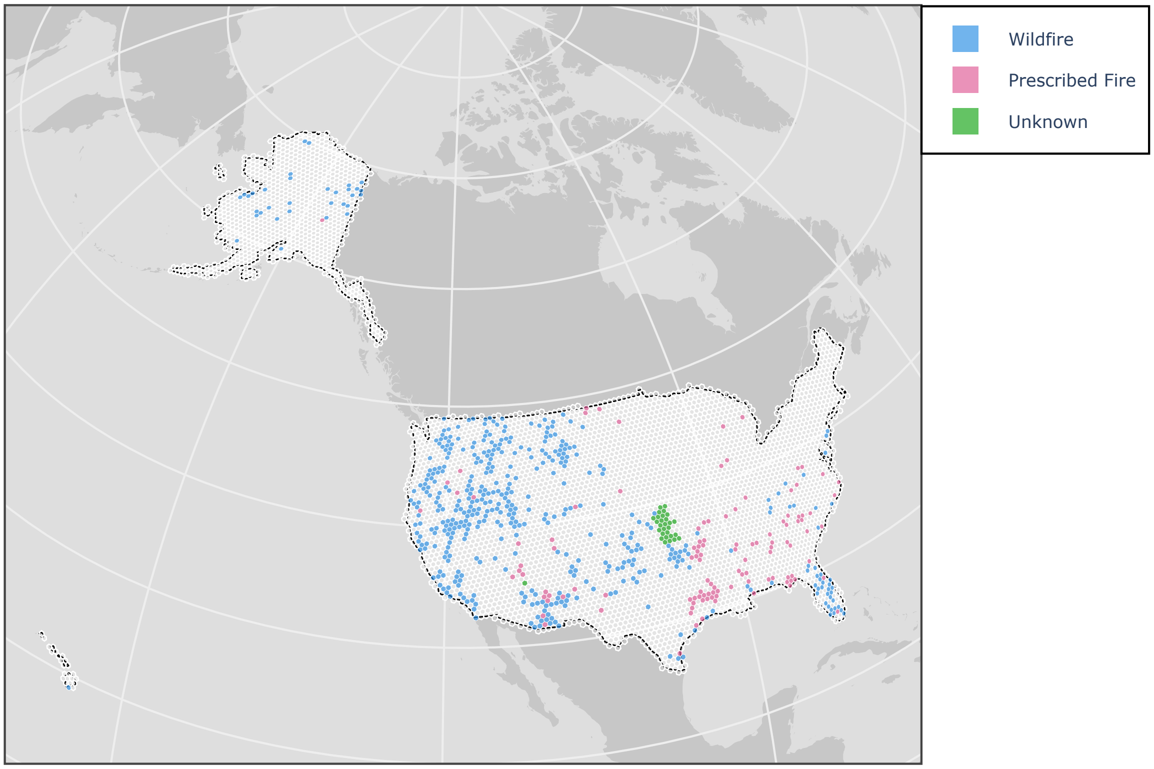


Figure 8: *Most frequent fire category by location (United States of America: 2017—hexagonal resolution = 4, average hexagon area = 11770 km^2).*

To make this visualization the required layer and its properties are passed via the `hexbin_layer` object.

```
{
  "hexbin_layer": {
    "data": "<data file location>",
    "hexbin_info": {
      "hex_resolution":4,
      "binning_field": "FIRE_TYPE",
      "binning_fn": "best"
    },
    "manager": {
      "marker": {
        "line": {
          "width": 0.1
        }
      },
      "colorscale": "Dark24"
    }
  },
```

The `hexbin_layer` object has 3 members being `data`, `hexbin_info`, and `manager`. Identical to the previous examples, the `data` member is now a full path to the data set containing the fire locations. Next, the hexbin_info member, which is also an object, specifies how the data is to be hexagonally binned. This is done through its 3 members: binning_field, binning_fn, and hex_resolution. Similar to Subsection 4.2, and Subsection 4.3, in this example the data is to be binned by incident location and the value displayed is the most frequent category of fire in each hexagon. The binning_field member of hexbin_info specifies that the display value be calculated from the FIRE_TYPE column, which is the column containing the category of fire. The binning_fn member of hexbin_info then specifies that the best option be selected as the display value (the most frequent value). The hex_resolution member of hexbin_info specifies the size of hexagon to be used. This shows that the hexagon size can also be specified as a member of the hexbin_info object unlike the previous examples. Finally, the manager specifies 2 properties that are passed into Plotly. The first member, marker is used to specify the width of the lines used for the hexagons in the hexbin layer. This is done through setting the width of the marker's line property; the same properties were set in Subsection 4.3. When set, the second member, colorscale specifies the colour scale to be used within the plot; in this case the colour scale is set to Dark24. This property was also set in Subsection 4.1.

Since the region of focus in this example is USA, a region layer containing the USA landmass is added to the plot via the regions object.

```
"regions": {
  "sample_Region_USA": {
    "data": "UNITED STATES OF AMERICA"
  }
},
```

The sample_Region_USA is the object defining this region layer. The data member of the region layer is set to UNITED STATES OF AMERICA.

Now extended grid layers are added to fill the gaps within the data and form a continuous grid. Once again, this is done by adding the grids object to the JSON file.

```
"grids": {
  "sample_Grid_USA": {
    "data": "UNITED STATES OF AMERICA",
    "convex_simplify": true
  }
},
```

Since the data spans the United States of America, we declare a grid layer that also spans this region. This grid layer is defined through the object sample_Grid_USA, whose data member is set to UNITED STATES OF AMERICA.

Next, some properties of the legend are set for aesthetic purposes. The properties of the legend are stored within the internal figure's layout properties. In order to interact with the internal figure's layout, the figure_layout object is added to the JSON file. For the full list of properties that can be set for the figure's layout, see [55].

```
  "figure_layout": {
    "legend": {
      "x": 0.8043,
      "bordercolor": "black",
      "borderwidth": 1,
      "font": {
        "size": 8
      }
    }
  },
```

The properties of the legend are set by adding the legend member/object to the figure_layout object. The legend property controls the different features of the legend, such as width, legend item sizing, and legend title; for the full list of input options, see [56]. The legend object has four members which control positioning x, the colour of the border bordercolor, the width of the border borderwidth, and the size of the font (controlled through the size member of the font property).

To do this, the internal figure's geo layout (Plotly) needs to set; this is done via adding the figure_geos. For the full list of properties that can be set for the figure's geo layout, see [51].

Next, a set of functions are specified in the functions object of the JSON file.

```
  "functions": {
    "clip_layers": {
      "clip": "hexbin+grids",
      "to": "regions"
    },
    "adjust_focus": {
      "on": "hexbin",
      "buffer_lat": [0,15],
      "rot_buffer_lon": -8
    }
  },
```

The first function is the clip_layers, which specifies that hexbin and grid layers be clipped to region layers; this same function is used in Subsection 4.1, and Subsection 4.3. The second function is the adjust_focus, which specifies that the plot be focused on the hexbin layer (but slightly shifted); this same function is used in Subsection 4.1, and Subsection 4.2.

Finally, the output location of the visualization is specified through the output. This step is identical to the examples shown in Subsection 4.1, Subsection 4.2, and Subsection 4.3.

```
  "output": {
    "filepath": "<output path>",
    "crop_output": true
  }
}
```

The full JSON file is given below. The Python module translation of this JSON is given in Annex D.

```json
{
  "hexbin_layer": {
    "data": "<data file location>",
    "hexbin_info": {
      "hex_resolution":4,
      "binning_field": "FIRE_TYPE",
      "binning_fn": "best"
    },
    "manager": {
      "marker": {
        "line": {
          "width": 0.1
        }
      },
      "colorscale": "Dark24"
    }
  },
  "regions": {
    "sample_Region_USA": {
      "data": "UNITED STATES OF AMERICA"
    }
  },
  "grids": {
    "sample_Grid_USA": {
      "data": "UNITED STATES OF AMERICA",
      "convex_simplify": true
    }
  },
  "figure_layout": {
    "legend": {
      "x": 0.8043,
      "bordercolor": "black",
      "borderwidth": 1,
      "font": {
        "size": 8
      }
    }
  },
  "functions": {
    "clip_layers": {
      "clip": "hexbin+grids",
      "to": "regions"
    },
    "adjust_focus": {
      "on": "hexbin",
      "buffer_lat": [0,15],
      "rot_buffer_lon": -8
    }
  },
  "output": {
    "filepath": "<output path>",
    "crop_output": true
  }
}
```

5 Discussion

This section discusses three existing limitations of GeoHexViz as of this writing. Specifically, these issues are concerning: geometries near the poles and the 180th meridian; issues related to colour bars; and missing hexagons.

5.1 180th meridian issues

GeoHexViz uses the GeoJSON format to plot data sets. With GeoJSON comes difficulties when geometries cross the 180th meridian [57]. Different libraries interpret geometries differently, and hence geometries that cross the 180th meridian may be interpreted as wrapping around the globe and avoiding the meridian entirely. In GeoHexViz, hexagonal geometries are supplied via Uber H3, and hence this issue has been discussed with the its developers [58]. In this package a simple solution to the problem is implemented, in the future it would be best to provide a more robust solution. The solution implemented involves tracking geometries that cross the meridian, and shifting their coordinates, making all of the coordinates either positive or negative. The theory behind this solution is discussed in [57]. The solution that is used works generally, however, when hexagons containing either the north or south pole are present, the solution to the 180th meridian issue persists. In Figure 9, the SAR data set used in Subsection 4.1, is used. This time however, the geometries are not clipped to the region of Canada, and instead span the globe including incident locations that are near the North Pole. Using a hexagonal resolution of 2, the issue presents itself in the final visualization.

The issue appears to cause a colour that bleeds through the entire plot and leaves a hexagon (or hexagons) empty. In the final plot, this issue may or may not appear as it only occurs at certain angles of rotation. Increasing the hexagonal resolution solves this issue for this example—an increase from hexagonal resolution 2 to hexagonal resolution 3 solves the issue for this example—and most others, but it should be investigated further. This colour bleed-through issue has also been discussed on the Plotly community forum and can be seen in [59].

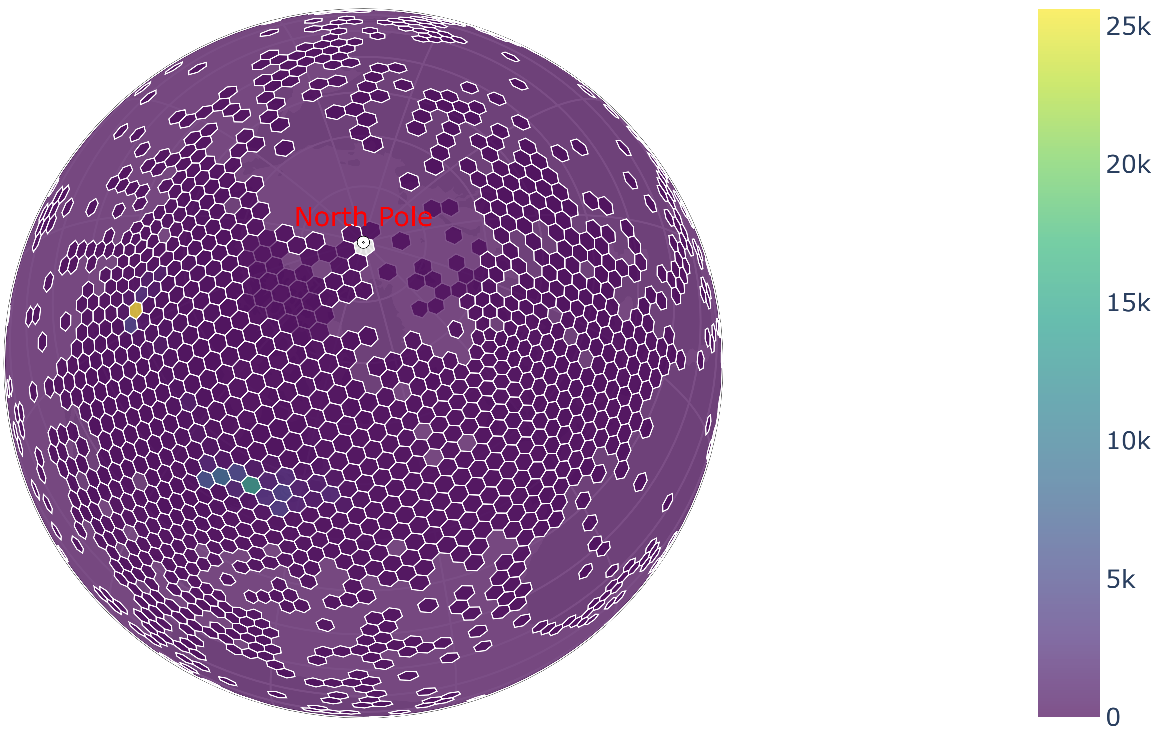


Figure 9: *180th meridian issue on North Pole (hexagonal resolution = 286,746 km^2).*

5.2 Colour bar issues

An issue related to the generation of discrete colour scales occurs under rare circumstances. These circumstances include generating discrete colour scales with not enough hues to fill the scale, and generating diverging discrete colour scales with the center hue in an incorrect position (not where the user specified). These issues have been noted and will be fixed in the near future.

In addition, there exists an issue with the positioning and height of the colour bar with respect to the plot area of the figure. When the dimensions of the plot area are not within a specific range of aspect ratios, the colour bar position and height may not be optimal. An example of this can be seen in Figure 10.

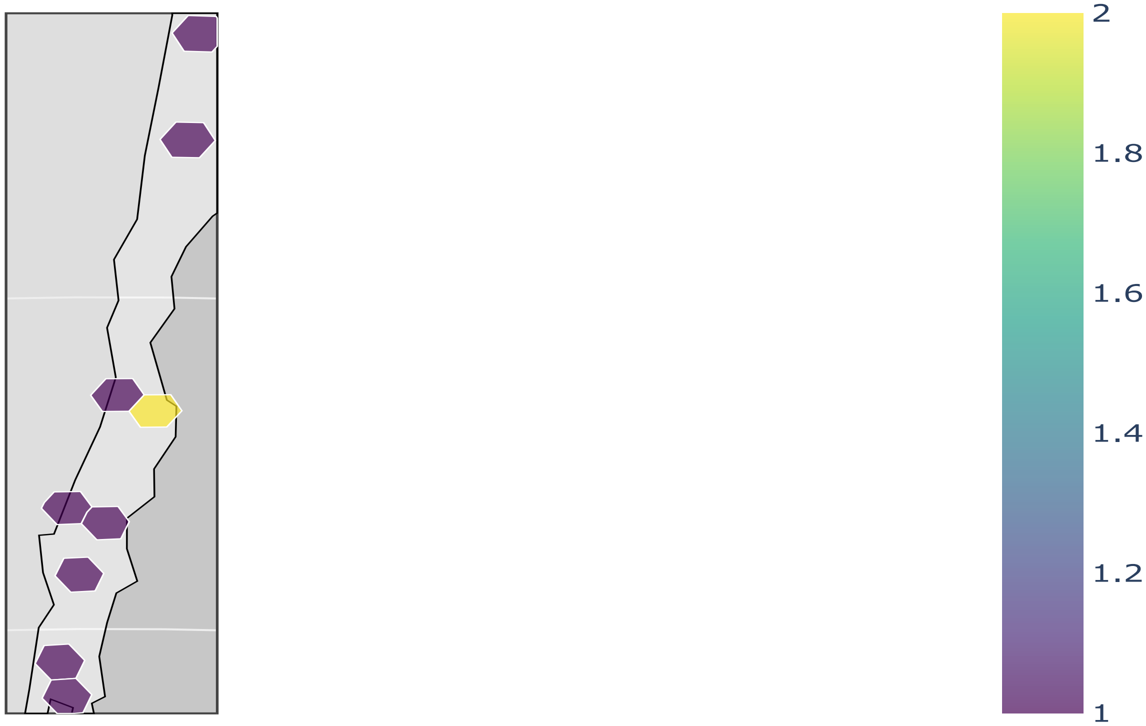


Figure 10: *Colour bar positioning issue (hexagonal resolution = 3, average hexagon area = 12392 km^2).*

Although the user is capable of altering the dimensions and positioning of the colour bar, this should be done automatically as it is a common feature of software that produces publication-quality choropleth maps. This issue has been discussed with some of the Plotly development team [60]. As this is an issue with the Plotly library itself, the library's developers have indicated that a calculation of plot area dimensions may be available in the future which would address in this issue.

5.3 Grid generation issues

GeoHexViz relies on the Python binding of the Uber H3 package in order to generate hexagons over polygons. This is done by passing the GeoJSON format of the polygon(s) to Uber H3. In some cases over large areas, grids may not generate properly. This error may manifest in GeoHexViz in the form of no hexagons, or multiple invalid hexagons being retrieved from Uber H3. This issue does not seem to be widely discussed, but in this document, an example is shown in Subsection 4.3. This is problematic for data sets that span large areas, and may result in required hexagons missing from the visualization.

6 Conclusion

For many analysts there exists a knowledge gap with respect GIS itself and its associated analytical techniques. The result is that it is both difficult and time-consuming for a such individuals to create geospatial visualizations. With the aim to help overcome this barrier, this document described GeoHexViz which provides a simple yet flexible approach to generating publication-quality geospatial visualizations of hexagonally binned data. GeoHexViz provides two mechanisms to do so: first, through a command-line script that reads a JSON file which specifies the visualization's properties; and second through invoking a series of Python functions provided by GeoHexViz within a user's own Python module. In addition, there exists many ways that an analyst may adjust the generated visualization if necessary via functions packaged with GeoHexViz or by passing arguments through GeoHexViz to the underlying libraries.

As of this publication, there exists three outstanding issues with GeoHexViz which are related to its underlying libraries: issues with Plotly regarding geometries that cross the 180th meridian; issues with Plotly regarding the positioning and height of the colour bar; and issues with Uber H3 regarding the generation of hexagons over large areas. In each case, once the underlying issue is addressed within the respective Python library, it is expected to be resolved within GeoHexViz.

References

[1] Simmons, S. (2018), 1.09 - Metadata and Spatial Data Infrastructure, In Huang, B., (Ed.), *Comprehensive Geographic Information Systems*, pp. 110–124, Oxford: Elsevier.

[2] Mostak, T. (2021), Geospatial - A Complete Introduction (online), https://www.omnisci.com/learn/geospatial (Access Date: July 2021).

[3] Ayalasomayajula, V. (2017), 7 techniques to VISUALIZE geospatial data (online), KDnuggets, https://www.kdnuggets.com/2017/10/7-techniques-visualize-geospatial-data.html (Access Date: August 2021).

[4] Feibush, E., Gagvani, N., and Williams, D. (2000), Visualization for situational awareness, *IEEE Computer Graphics and Applications*, 20(5), 38–45.

[5] Laskey, K. B., Wright, E. J., and da Costa, P. C. (2010), Envisioning uncertainty in geospatial information, *International Journal of Approximate Reasoning*, 51(2), 209–223. Bayesian Model Views.

[6] Kovařík, V. (2011), Possibilities of geospatial data analysis using spatial modeling in ERDAS IMAGINE, In *Proceedings of the International Conference on Military Technologies 2011-ICMT'11*, pp. 1307–1313.

[7] Connable, B., Perry, W. L., Doll, A., Lander, N., and Madden, D. (2014), Modeling, Simulation, and Operations Analysis in Afghanistan and Iraq: Operational Vignettes, Lessons Learned, and a Survey of Selected Efforts, RAND Corporation.

[8] Goodrich, D. C., Heilman, P., Guertin, D., Levick, L. R., Burns, I., Armendariz, G., and Wei, H. (2019), Automated geospatial watershed assessment (AGWA) to aid in sustaining military mission and training, (Technical Report) USDA-ARS Southwest Watershed Research Center (SWRC) Tucson United States.

[9] Hunter, G., Chan, J., and Rempel, M. (2021), Assessing the Impact of Infrastructure on Arctic Operations, Defence Research and Development Canada, Scientific Report DRDC-RDDC-2021-R024.

[10] Vartak, M., Madden, S., Parameswaran, A., and Polyzotis, N. (2014), SeeDB: Automatically Generating Query Visualizations, *Proc. VLDB Endow.*, 7(13), 1581–1584.

[11] Bertazzon, S. (2013), Rethinking GIS teaching to bridge the gap between technical skills and geographic knowledge, *Journal of Research and Didactics in Geography*, 1, 67–72.

[12] Jack Dangermond, L. D. (2021), ArcGIS Online (online), https://www.arcgis.com/ (Access Date: May 2021).

[13] Sherman, G. (2021), QGIS - A Free and Open Source Geographic Information System (online), https://qgis.org/en/site/ (Access Date: May 2021).

[14] Bostock, M. (2021), D3.js - Data-Driven Documents (online), https://d3js.org/ (Access Date: May 2021).

[15] GISGeography (2021), ArcGIS Review: Is ArcMap the Best GIS Software? (online), https://gisgeography.com/esri-arcgis-software-review-guide/ (Access Date: August 2021).

[16] GrindGIS (2021), Pros and Cons of QGIS (online), https://grindgis.com/software/pros-and-cons-of-qgis (Access Date: May 2021).

[17] Cook, P. (2021), D3 in Depth: Geographic (online), https://www.d3indepth.com/geographic/ (Access Date: May 2021).

[18] Plotly (2021), Plotly Python Open Source Graphing Library (online), Plotly, https://plotly.com/python/ (Access Date: March 2021).

[19] Sipe, N. G. and Dale, P. (2003), Challenges in using geographic information systems (GIS) to understand and control malaria in Indonesia, *Malaria Journal*.

[20] Briney, A. (2014), Binning in GIS (online), GIS Lounge, https://www.gislounge.com/binning-gis/ (Access Date: March 2021).

[21] Field, K. (2012), Using a binning technique for point-based multiscale web maps (online), ArcGIS Online, https://www.esri.com/arcgis-blog/products/arcgis-online/mapping/using-a-binning-technique-for-point-based-multiscale-web-maps/ (Access Date: April 2021).

[22] Sinha, A. (2019), Spatial Modelling Tidbits: Honeycomb or Fishnets? (online), Towards Data Science, https://towardsdatascience.com/spatial-modelling-tidbits-honeycomb-or-fishnets-7f0b19273aab (Access Date: March 2021).

[23] Birch, C., Oom, S., and Beecham, J. (2007), Rectangular and hexagonal grids used for observation, experiment and simulation in ecology, *Ecological Modelling*, 206, 347–359.

[24] Battersby, S. E., Strebe, D., and Finn, M. P. (2017), Shapes on a plane: evaluating the impact of projection distortion on spatial binning, *Cartography and Geographic Information Science*, 44(5), 410–421.

[25] McKinney, W. (2021), pandas.DataFrame (online), https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html (Access Date: March 2021).

[26] Jordahl, K. (2021), geopandas.GeoDataFrame (online), https://geopandas.org/docs/reference/api/geopandas.GeoDataFrame.html (Access Date: March 2021).

[27] Caliper (2021), What is a layer? (online), Caliper Mapping and Transportation Glossary, https://www.caliper.com/glossary/what-is-a-map-layer.htm (Access Date: October 2021).

[28] Brodsky, I. (2021), H3 Documentation (online), Towards Data Science, https://h3geo.org/docs (Access Date: March 2021).

[29] Jordahl, K. (2021), GeoPandas (0.9.0) (online), https://geopandas.org/index.html (Access Date: March 2021).

[30] McKinney, W. (2021), Pandas Documentation (1.2.3) (online), https://pandas.pydata.org/docs/ (Access Date: March 2021).

[31] Gilles, S. (2013), The Shapely User Manual (1.7.1) (online), Waterloo Maple Inc, https://shapely.readthedocs.io/en/stable/manual.html (Access Date: April 2021).

[32] Plotly (2021), Choropleth Traces (online), https://plotly.com/python/reference/choropleth (Access Date: May 2021).

[33] Plotly (2021), Scattergeo Traces (online), https://plotly.com/python/reference/scattergeo (Access Date: May 2021).

[34] JSON (2021), Introducing JSON (online), https://www.json.org/json-en.html (Access Date: October 2021).

[35] Search and Rescue Mission Management System (2020), Search and Rescue Incidents. Joint Rescue Coordination Centre (JRCC). downloaded by Capt. David Burneau for CJOC SAR on January 20th, 2020.

[36] Plotly (2021), Built-in Continuous Color Scales in Python (online), Plotly — Graphing Libraries, https://plotly.com/python/builtin-colorscales/ (Access Date: 18 October 2021).

[37] Plotly (2021), Discrete Colors in Python (online), Plotly — Graphing Libraries, https://plotly.com/python/discrete-color/ (Access Date: 18 October 2021).

[38] Plotly (2021), Choropleth Traces (Colorscale) (online), Plotly — Graphing Libraries, https://plotly.com/python/reference/choropleth/#choropleth-colorscale (Access Date: 18 October 2021).

[39] Plotly (2021), Choropleth Traces (Colorbar) (online), Plotly — Graphing Libraries, https://plotly.com/python/reference/choropleth/#choropleth-colorbar (Access Date: 18 October 2021).

[40] NaturalEarth (2021), Natural Earth—Free vector and raster map data at 1:10m, 1:50m, and 1:110m scales (online), https://www.naturalearthdata.com/ (Access Date: September 2021).

[41] pdfCropMargins (2021), pdfCropMargins 1.0.5 (online), PyPi, https://pypi.org/project/pdfCropMargins/ (Access Date: October 2021).

[42] GVA (2021), Gun Violence Archive (online), gunviolencearchive, https://www.gunviolencearchive.org/reports/mass-shooting (Access Date: October 2021).

[43] Plotly (2021), Python Figure Reference: scattergeo Traces — textposition (online), Plotly Graphing Libraries, https://plotly.com/python/reference/scattergeo/#scattergeo-textposition (Access Date: 20 October 2021).

[44] Plotly (2021), Python Figure Reference: scattergeo Traces — marker (online), Plotly Graphing Libraries, https://plotly.com/python/reference/scattergeo/#scattergeo-marker (Access Date: 20 October 2021).

[45] Plotly (2021), Styling Markers in Python (online), Plotly Graphing Libraries, https://plotly.com/python/marker-style/ (Access Date: 20 October 2021).

[46] Plotly (2021), Python Figure Reference: scattergeo Traces — marker symbol (online), Plotly Graphing Libraries, https://plotly.com/python/reference/scattergeo/#scattergeo-marker-symbol (Access Date: 20 October 2021).

[47] Plotly (2021), Python Figure Reference: scattergeo Traces — marker symbol (online), Plotly Graphing Libraries, https://plotly.com/python/reference/scattergeo/#scattergeo-marker-size (Access Date: October 2021).

[48] Plotly (2021), Python Figure Reference: scattergeo Traces — marker line (online), Plotly Graphing Libraries, https://plotly.com/python/reference/scattergeo/#scattergeo-marker-line (Access Date: October 2021).

[49] Larion, A. (2016), Aerial Bombing Operations in World War II (online), Kaggle, https://www.kaggle.com/usaf/world-war-ii?select=operations.csv (Access Date: September 2021).

[50] Plotly (2021), Choropleth Traces (Marker) (online), Plotly — Graphing Libraries, https://plotly.com/python/reference/choropleth/#choropleth-marker (Access Date: 18 October 2021).

[51] Plotly (2021), Plotly graph_objects package: layout geo (online), Plotly Graphing Libraries, https://plotly.com/python-api-reference/generated/plotly.graph_objects.layout.geo.html#module-plotly.graph_objects.layout.geo (Access Date: 18 October 2021).

[52] Plotly (2021), Plotly graph_objects package: layout geo lonaxis (online), Plotly Graphing Libraries, https://plotly.com/python-api-reference/generated/plotly.graph_objects.layout.geo.html#plotly.graph_objects.layout.geo.Lonaxis (Access Date: 18 October 2021).

[53] Plotly (2021), Plotly graph_objects package: layout geo projection (online), Plotly Graphing Libraries, https://plotly.com/python-api-reference/generated/plotly.graph_objects.layout.geo.html#plotly.graph_objects.layout.geo.Projection (Access Date: 18 October 2021).

[54] MBTS (2021), Monitoring Trends in Burn Severity Burned Area Boundaries (Feature Layer) (online), ArcGIS, https://hub.arcgis.com/datasets/usfs::monitoring-trends-in-burn-severity-burned-area-boundaries-feature-layer/about (Access Date: September 2021).

[55] Plotly (2021), Plotly graph_objects package: layout (online), Plotly Graphing Libraries, https://plotly.com/python-api-reference/generated/plotly.graph_objects.Layout.html#plotly.graph_objects.Layout (Access Date: October 2021).

[56] Plotly (2021), Plotly graph_objects package: layout legend (online), Plotly Graphing Libraries, https://plotly.com/python-api-reference/generated/plotly.graph_objects.layout.html#plotly.graph_objects.layout.Legend (Access Date: October 2021).

[57] MacWright, T. (2016), The 180th Meridian (online), https://macwright.com/2016/09/26/the-180th-meridian.html (Access Date: April 2021).

[58] Abou Zeidan, T. (2021), Q: Invalidity of Polygons in GeoJSON, GeoPandas (online), https://github.com/uber/h3-py/issues/187 (Access Date: July 2021).

[59] Abou Zeidan, T. (2021), Color Bleedthrough Invalid Polygon or GeoJSON (online), https://community.plotly.com/t/color-bleedthrough-invalid-polygon-or-geojson/52549 (Access Date: July 2021).

[60] Abou Zeidan, T. (2021), [Feature Request | Bug Report] Plot Area / Colorbar Size Variance (Geos) (online), https://github.com/plotly/plotly.py/issues/3288 (Access Date: July 2021).

Annex A Search and Rescue—Python module input

This annex lists the Python code to create Figure 5.

```
from geohexviz.builder import PlotBuilder

myBuilder = PlotBuilder()

# set hexbin layer
myBuilder.set_hexbin(
    data="<path to data.csv>",
    hex_resolution=4,
    manager=dict(
        colorscale="Viridis",
        colorbar=dict(
            x=0.8325
        )
    )
)

# add region layers
myBuilder.add_region(
    name="sample_Region_CANADA",
    data="CANADA"
)

# add grid layers
myBuilder.add_grid(
    name="sample_Grid_CANADA",
    data="CANADA"
)

# invoke functions
myBuilder.clip_layers(
    clip="hexbin+grids",
    to="regions"
)
myBuilder.adjust_focus(
    on="regions",
    buffer_lat=[0, 3]
)
myBuilder.logify_scale(
    exp_type="r"
)

# finalize and output
myBuilder.finalize()
myBuilder.output(
    filepath="<path to output (.pdf)>",
    crop_output=True
)
```

Annex B Mass shootings—Python module input

This annex lists the Python code to create Figure 6.

```python
from geohexviz.builder import PlotBuilder

myBuilder = PlotBuilder()

# set hexbin layer
myBuilder.set_hexbin(
    data="<data file location>",
    hex_resolution=3,
    hexbin_info=dict(
        binning_field="killed_injured",
        binning_fn="sum"
    ),
    manager=dict(
        colorbar=dict(
            x=0.8365
        )
    )
)

# add region layers
myBuilder.add_region(
    name="sample_Region_USA",
    data="UNITED STATES OF AMERICA"
)

myBuilder.add_point(
    name="sample_Point_EPICENTERS",
    data="<epicenters file location>",
    manager=dict(
        textposition=[
            "top center",
                        "top center",
                        "middle right",
                        "top center",
                        "top left",
                        "bottom right",
                        "top center",
                        "top center",
                        "top center",
                        "top center"
        ],
        marker=dict(
            symbol="square-dot",
            size=4,
            line=dict(
                width=0.5
            )
        )
    )
)

# invoke functions
myBuilder.remove_empties()
myBuilder.adjust_focus(
    on="hexbin+grids",
    buffer_lat=[0, 15],
    rot_buffer_lon=-8
```

```
)
myBuilder.logify_scale(
    exp_type="r"
)

# finalize and output
myBuilder.finalize()
myBuilder.output(
    filepath="<output path>",
    crop_output=True
)
```

Annex C World War 2 bombings—Python module input

This annex lists the Python code to create Figure 7.

```
from geohexviz.builder import PlotBuilder

myBuilder = PlotBuilder()

# set hexbin layer
myBuilder.set_hexbin(
    data="<data file location>",
    hexbin_info=dict(
        binning_field="high_explosives_weight_tons",
        binning_fn="sum"
    ),
    hex_resolution=4,
    manager=dict(
        marker=dict(
            line=dict(width=0.45)
        ),
        colorscale="Viridis",
        colorbar=dict(
            x=0.82
        )
    )
)

# add region layers
myBuilder.add_region(
    name="sample_Region_EUROPE",
    data="EUROPE"
)

# add grid layers
myBuilder.add_grid(
    name="sample_Grid_EUROPE",
    data="EUROPE",
    convex_simplify=True
)
myBuilder.add_grid(
    name="sample_Grid_RUSSIA",
    data="RUSSIA",
    convex_simplify=True
)

# update grid manager
myBuilder.update_grid_manager(
    marker=dict(
        line=dict(width=0.45)
    )
)

# update figure geos
myBuilder.update_figure(
    geos=dict(
        lataxis=dict(
            range=[35, 58]
        ),
        lonaxis=dict(
            range=[0, 43]
        ),
```

```
        projection=dict(
            rotation=dict(
                lat=46.63321662159487,
                lon=11.21560455920799
            )
        )
    )
)

# invoke functions
myBuilder.clip_layers(
    clip="hexbin+grids",
    to="regions"
)
myBuilder.logify_scale(
    exp_type="r"
)
# * Unlike JSON input mechanism, in a module adjust\_focus is not
# * invoked by default, the user has to invoke it

# finalize and output
myBuilder.finalize()
myBuilder.output(
    filepath="<output path>",
    crop_output=True
)
```

Annex D Forest fires—Python module input

This annex lists the Python code to create Figure 8.

```
from geohexviz.builder import PlotBuilder

myBuilder = PlotBuilder()

# set hexbin layer
myBuilder.set_hexbin(
    data="<data file location>",
    hexbin_info=dict(
        hex_resolution=4,
        binning_field="FIRE_TYPE",
        binning_fn="best"
    ),
    manager=dict(
        marker=dict(
            line=dict(
                width=0.1
            )
        ),
        colorscale="Dark24"
    )
)

# add region layers
myBuilder.add_region(
    name="sample_Region_USA",
    data="UNITED STATES OF AMERICA"
)

# add grid layers
myBuilder.add_grid(
    name="sample_Grid_USA",
    data="UNITED STATES OF AMERICA"
)

# alter figure layout
myBuilder.update_figure(
    layout=dict(
        legend=dict(
            x=0.8043,
            bordercolor="black",
            borderwidth=1,
            font=dict(
                size=8
            )
        )
    )
)

# invoke functions
myBuilder.clip_layers(
    clip="hexbin+grids",
    to="regions"
)
myBuilder.adjust_focus(
    on="hexbin+grids",
    buffer_lat=[0, 15],
    rot_buffer_lon=-8
```

```
)

# finalize and output
myBuilder.finalize()
myBuilder.output(
    filepath="<output path>",
    crop_output=True
)
```

Acronyms and abbreviations

GIS Geographic Information System

SAR search and rescue

OR&A Operations Research and Analysis

.csv comma-separated values

SMSS Search and Recue Mission Management System

JSON JavaScript Object Notation

DOCUMENT CONTROL DATA

*Security markings for the title, abstract and keywords must be entered when the document is sensitive.

1. ORIGINATOR (The name and address of the organization preparing the document. A DRDC Centre sponsoring a contractor's report, or a tasking agency, is entered in Section 8.) DRDC – Centre for Operational Research and Analysis NDHQ Carling, 60 Moodie Drive, building 7S.2, Ottawa ON K1A 0K2, Canada	2a. SECURITY MARKING (Overall security marking of the document, including supplemental markings if applicable.) CAN UNCLASSIFIED 2b. CONTROLLED GOODS NON-CONTROLLED GOODS DMC A

3. TITLE (The document title and subtitle as indicated on the title page.)

GeoHexViz—Geospatial visualization using hexagonal binning software: Design reference and instruction manual

4. AUTHORS (Last name, followed by initials – ranks, titles, etc. not to be used. Use semi-colon as delimiter.)

Zeidan, T. A.; Rempel, M.

5. DATE OF PUBLICATION (Month and year of publication of document.) December 2021	6a. NO. OF PAGES (Total pages, including Annexes, excluding DCD, covering and verso pages.) 57	6b. NO. OF REFS (Total cited in document.) 60

7. DOCUMENT CATEGORY (e.g., Scientific Report, Contract Report, Scientific Letter)

Reference Document

8. SPONSORING CENTRE (The name and address of the department project or laboratory sponsoring the research and development.)

DRDC – Centre for Operational Research and Analysis
NDHQ Carling, 60 Moodie Drive, building 7S.2, Ottawa ON K1A 0K2, Canada

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.) 06ac	9b. CONTRACT NO. (If appropriate, the applicable contract number under which the document was written.)
10a. DRDC PUBLICATION NUMBER DRDC-RDDC-2021-D183	10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned to this document either by the originator or by the sponsor.)

11a. FUTURE DISTRIBUTION WITHIN CANADA (Approval for further dissemination of the document. Security classification must also be considered.)

Public release

11b. FUTURE DISTRIBUTION OUTSIDE CANADA (Approval for further dissemination of the document. Security classification must also be considered.)

Public release

12. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Use semi-colon as a delimiter.)

visualization; python; hexagonal binning

13. ABSTRACT/RÉSUMÉ (When available in the document, the French version of the abstract must be included here.)

Geospatial visualization is an important communication method that is often used in military operations research to convey analyses to both analysts and decision makers. When these types of visualizations include a large amount of point-like data, binning—in particular, hexagonal binning—may be used to summarize the data and subsequently produce an effective visualization. However, creating such visualizations may be frustrating for many since it requires in-depth knowledge of both Geographic Information Systems and analytical techniques, not to mention access to software that may require a paid license, training, and perhaps knowledge of a programming language. In this document we describe GeoHexViz which aims to reduce the time, in-depth knowledge, and programming required to produce publication-quality geospatial visualizations that use hexagonal binning. We describe the high-level design of GeoHexViz, its functional specification, and present four examples that demonstrate the capabilities of GeoHexViz in action. For each, we describe the two methods that GeoHexViz provides to do so: first, a command-line script whose input is a JavaScript Object Notation file that contains the visualization's properties; and second, a Python script that imports and invokes functions found in the software's Python modules.

La visualisation géospatiale est une méthode de communication importante, souvent utilisée pour transmettre des analyses aux analystes et aux décideurs participants à la recherche sur les opérations militaires. Lorsque ces types de visualisation englobent une grande quantité de données ponctuelles, le groupement de données par classe — et plus particulièrement le groupement de données par classe hexagonale — peut être utilisé pour synthétiser les données et ainsi générer une visualisation efficace. Cependant, la création de telles visualisations peut être contrariante puisqu'elle suppose une connaissance approfondie des systèmes d'information géographique et des techniques d'analyse, sans oublier l'accès à un logiciel (sous licence ou pas), une formation et la maîtrise d'un langage de programmation. Dans le présent document, nous vous présentons GeoHexViz, une ressource qui permet de réduire le temps, les connaissances et la programmation nécessaires à la production de visualisations géospatiales diffusables basées sur des groupements de données par classe hexagonale. Nous décrivons la conception de haut niveau et la spécification fonctionnelle de GeoHexViz, et présentons quatre exemples qui illustrent les capacités de GeoHexViz. Pour chacun de ces exemples, nous expliquons les deux méthodes proposées par GeoHexViz : 1) un script de commandes dont le fichier d'entrée est un fichier de notation objet JavaScript (JSON) qui contient les propriétés de la visualisation ; 2) un script Python qui importe des fonctions trouvées dans les modules Python du logiciel et les exécute.