```
// Ray-casting with two transfer functions
def render(volume, x, y):
        step = 1
        line_iter = orthogonal_iter(volume, x, y, step)
        color = make_float4(0)
        tcol1 = make_float4(0)
        tcol2 = make_float4(0)
        val = make_float2(0)
        for elem in line_iter:
                val = linear_query_3d(volume, elem)
                tcol1 = transfer(val.x,1)
                tcol2 = transfer(val.y,2)
                tcol.xyz = (tcol1.xyz*tcol1.w + tcol2.xyz*tcol2.w)
                           /(tcol1.w + tcol2.w)
                tcol.w = max(tcol1.w, tcol2.w)
        // alpha compositing
        color = alpha_compositing(color, tcol)
        if color.w > 254: break
        return RGBA(color)

// Merging output buffers (Sort-Last)
def composite(front, back, x, y):
         a = point_query_2d(front, x, y)
         b = point_query_2d(back, x, y)
         c = alpha_compositing(a, b)
         return RGBA(c)

def main():
        volume = load_data_3d('zebrafish.dat', out_of_core=True)
        enable_viewer(render(volume,x,y).range(x=0:1024,y=0:1024)
                                        .dtype(volume, uchar)
                                        .split(volume, z=4)
                                        .merge(composite,'front-to-back')
                                        .halo(volume,1) ,'TFF2', '3D', 256)
```

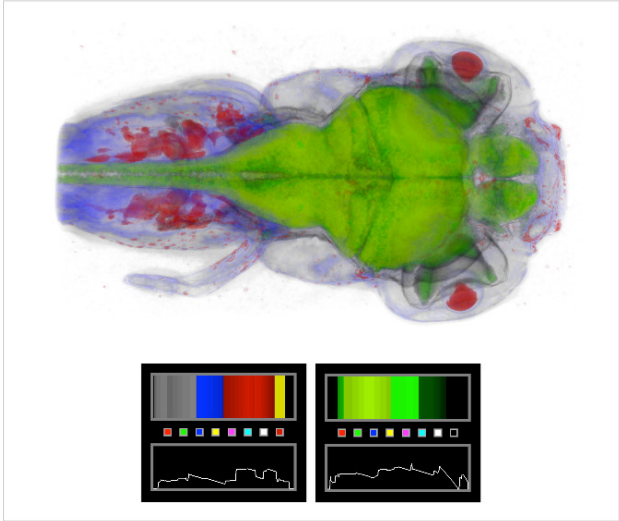

Figure 3.2 – Vivaldi code and output (Extracted from [CCQ+er]).

3.2 ViSlang

ViSlang [RBGH14] is an interpreted DSL elaborated to offer the possibility of creating scientific visualizations for facilitating its creation by scientists and avoiding the difficulties of handling a low-level language. A great contribution offered by ViSlang is the possibility of extending the DSL using *slangs*. This feature allows the implementation of others DSLs to address other issues that are within the domain. Figure 3.3 illustrates the use of *slangs*.

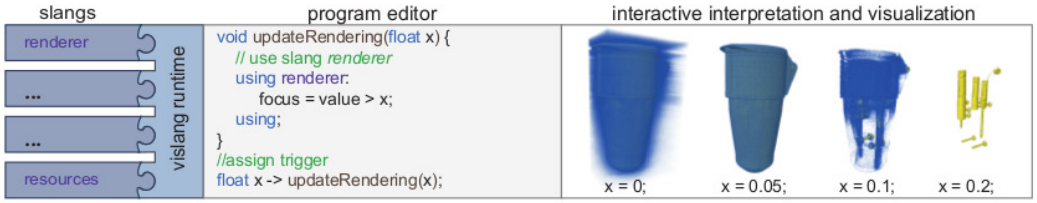

Figure 3.3 – Slangs in ViSlang (Extracted from [RBGH14]).

ViSlang is a system that decreases the cost of developing new DSLs since it integrates multiple DSLs in one solution to support their flexibility. Figure 3.3 provides an overview of ViSlang design. ViSlang has a library and an execution environment with support for an extension mechanism. By integrating it into a visualization system, the runtime can execute commands and change the behavior of the visualization. This runtime performs as a unified programming interface. Internally, DSLs that extend ViSlang are named slangs. It works in the following mode. User input is received and executed by an interpreter. If a command starts with the keyword *using*, it will be transmitted to the corresponding slang. Figure 3.3 demonstrates an example code that makes use of the slang renderer. It offers

a DSL that allows properties of data mapping to visual elements. This example gives an overview of the ViSlang runtime system where multiple DSLs are combined using common data processing and visualization modules. ViSlang currently focuses on processing and visualization of static volumes. This is not a restriction of the system design but the current implementation. However, one of the major design goals of ViSlang is its extensibility and ability to support other data structures.

3.3 Diderot

Diderot [CKR+12, KCS+16] is a DSL that simplifies portable implementation of parallel methods of biomedical image analysis and visualization. Image analysis extracts quantitative or geometric descriptions of the image structure to characterize specific properties of the underlying organ or tissue. Visualization combines measurements of local image data properties with elements of computer graphics to qualitatively depict structures via rendered images. Diderot supports a high-level model of computation that supports continuous tensor fields.

Diderot permits programmers to express algorithms directly regarding tensors, tensor fields, and tensor field operations, using the same mathematical notation that is used in traditional vector and tensor calculus. Diderot objective is to be useful for prototyping image analysis and visualization methods in contexts where a meaningful evaluation of the methods requires its application to real image data. Besides, the real data volumes are of a size that requires efficient parallel computation. With its support for high-level mathematical notation, Diderot is also useful in educational contexts where the conceptual transparency of the implementation is of primary importance.

Figure 3.4 presents an example of a simple, direct volume rendering code. This example code also demonstrates a field to implement a color assignment function (*i.e.*, the RGB field). This example shows this using bilinear interpolation, which is provided by the kernel. The output image from this code also includes an image of the bivariate colormap function.

The DSL compiler has approximately 19,000 lines of code that are organized in three phases: the front-end, optimization and reducing, and code generation. Different target languages perform code generation: sequential C code with vector extensions, parallel C code, OpenCL, and CUDA.

```
visualization: clusteredmap;
settings {
  latitude: field 7;
  longitude: field 8;
  marker-text: field 1 field 2 field 3 field 4;
  page-title: "Airports in World";
  size: full;
}
data {
  file: "vis_codes/airports.dat";
  structure {
    delimiter: ',';
    end-register: newline;
  }
}
```

Listing 4.1 – Airports in world (clustered map) code.

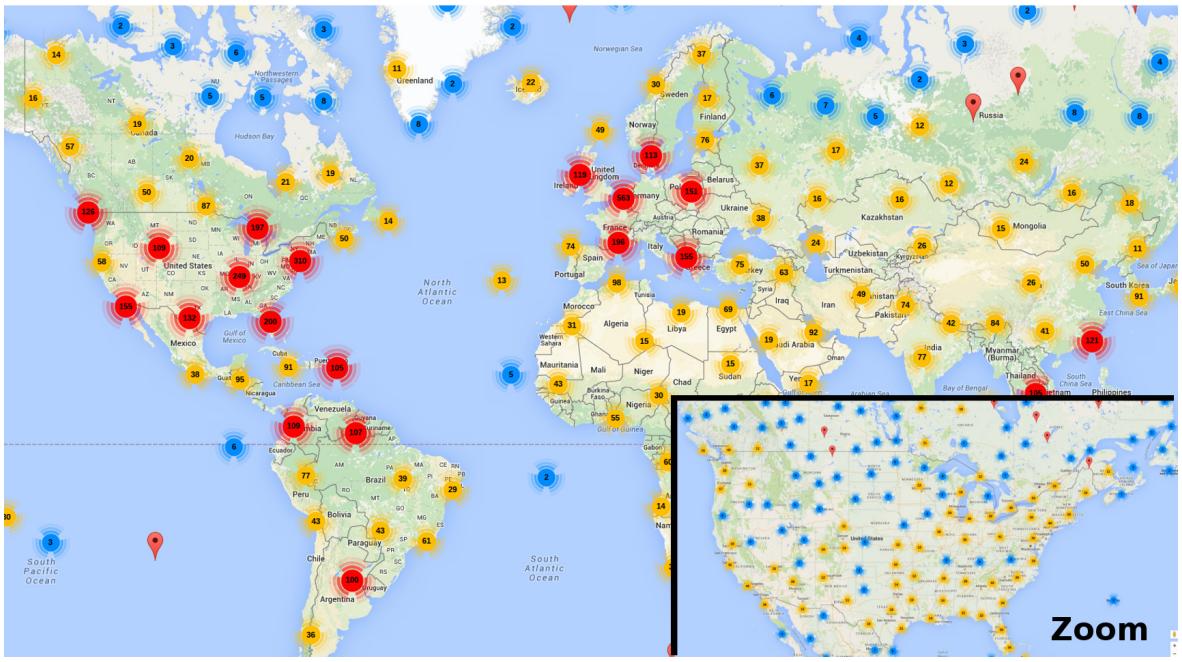

Figure 4.8 – Airports in world (clustered map).

Flickr Photos with "Computer" Word as Tag (clustered map)

This visualization also used clustered map to display markers that represent a photo taken and uploaded to Flickr. This visualization used a filter to select just pictures that have a tag that contains the word `computer`. This visualization displayed more occurrences of pictures with this tag in the United States and England. Figure 4.9 presents the generated visualization. Listing 4.2 describes the code for creating this visualization using GMaVis.

```
visualization: clusteredmap;
settings {
  latitude: field 12;
  longitude: field 11;
  marker-text: field 9 "and" field 8;
  page-title: "Flicker photos with computer tag in 2014";
  size: full;
}
data {
  file: "yfcc100m_dataset-0";
  structure {
    delimiter: tab;
    end-register: newline;
    date-format: "YYYY-MM-DD";
  }
  filter:  field 8 contains "computer" or field 9 contains "computer";
}
```

Listing 4.2 – Flickr photos with "Computer" word as tag (Clustered Map) code.

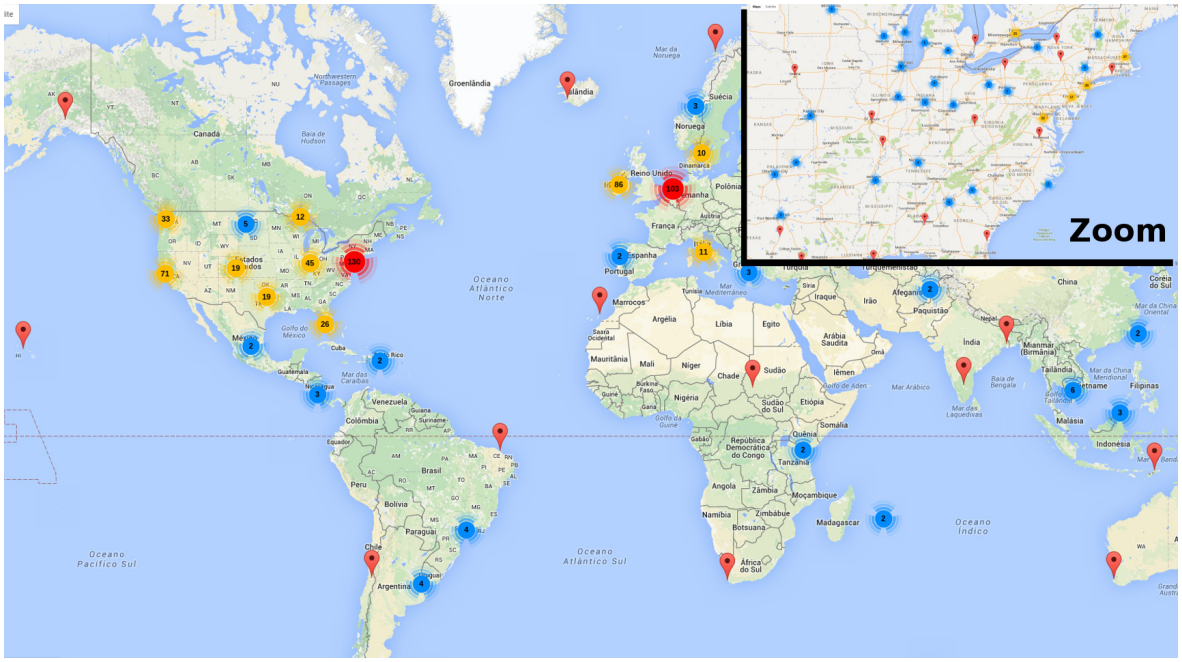

Figure 4.9 – Flickr photos with "Computer" word as tag (clustered map).

Airports in world (heatmap).

In this visualization, a marker is displayed for each airport in the world. Here a heatmap visualization was used. This visualization highlight places that have higher data frequency. In this case, it is possible to verify a greater number of airports in Europe. Figure 4.10 presents the generated visualization. Listing 4.3 describes the code for creating this visualization using GMaVis.

```
visualization: heatmap;
settings {
  latitude: field 7;
  longitude: field 8;
  marker-text: field 1 field 2 field 3 field 4;
  page-title: "Airports in World";
  size: full;
}
data {
  file: "vis_codes/airports.dat";
  structure {
    delimiter: ',';
    end-register: newline;
  }
}
```

Listing 4.3 – Airports in world (heatmap) code.

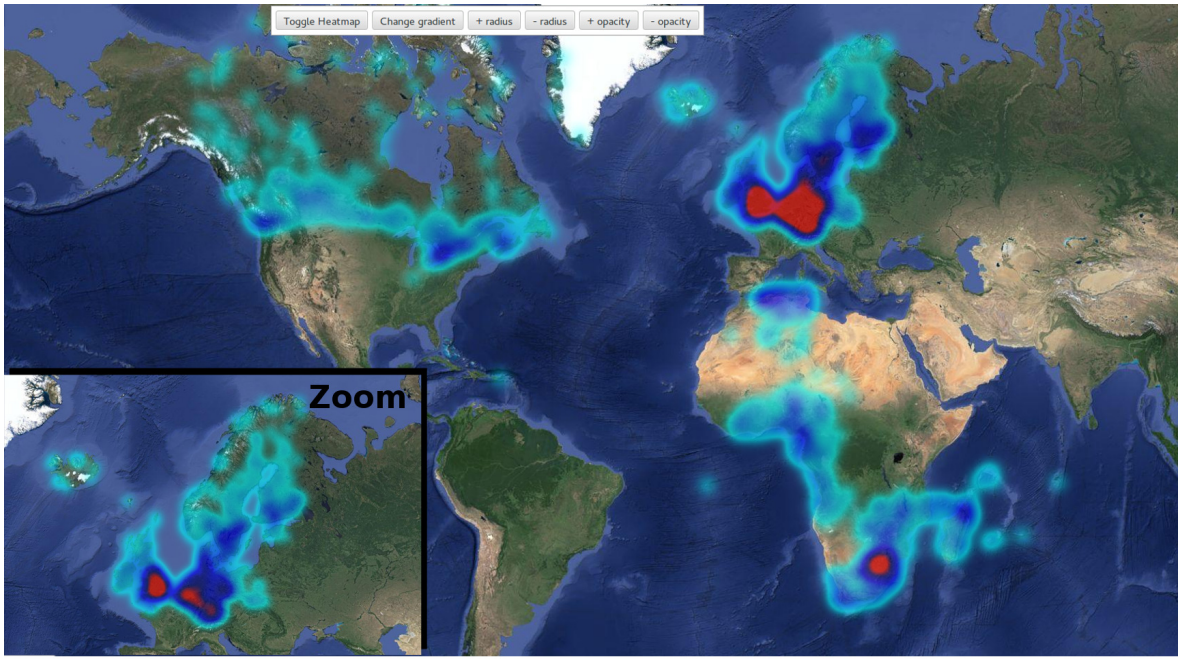

Figure 4.10 – Airports in world (heatmap).

Traffic Accidents in Porto Alegre, Brazil (heatmap).

This visualization used 2013 traffic accident data to display areas with greater frequency of accidents. In this visualization, areas with the color red have more accidents than those in yellow or green. Figure 4.11 presents the visualization generated. Listing 4.4 describes the code for creating this visualization using GMaVis.

```
visualization: heatmap;
settings {
  latitude: field 42;
  longitude: field 41;
  page-title: "Acidentes Poa 2014";
  size: full;
  zoom-level: 11;
}
data {
  h-file: "acidentes-2013.csv";
  structure {
    delimiter: ';';
    end-register: newline;
    date-format: "YYYY-MM-DD";
  }

  filter: field 13 is greater than 0 or field 14 is greater than 0;
}
```

Listing 4.4 – Traffic accidents in Porto Alegre, Brazil (heatmap) code.

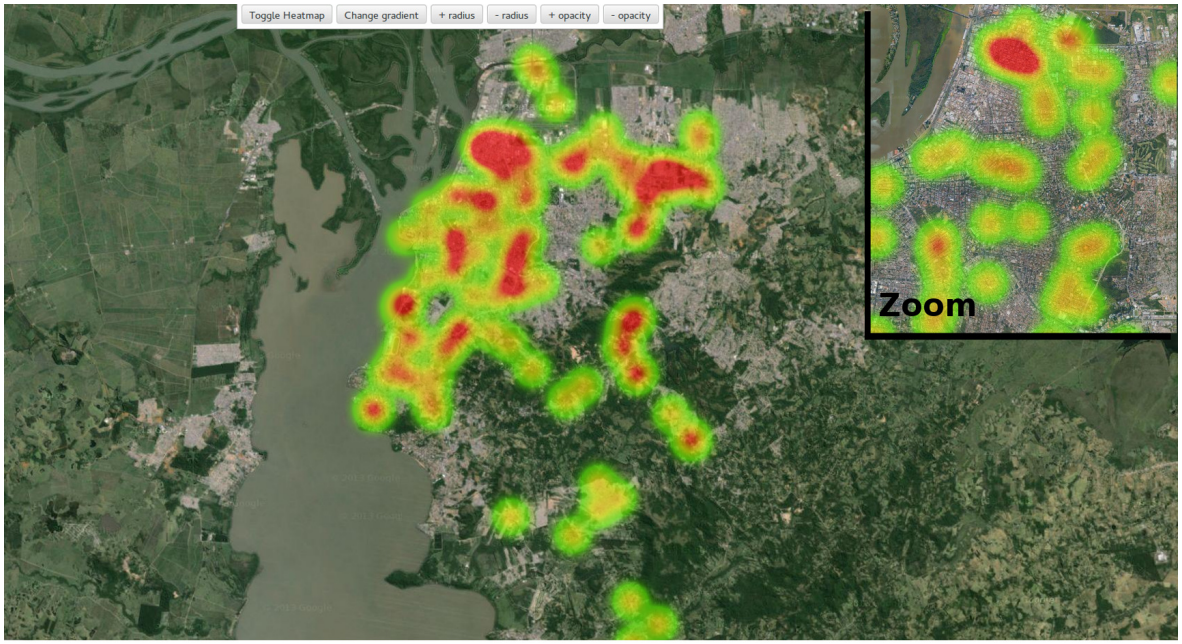

Figure 4.11 – Traffic accidents in Porto Alegre, Brazil (heatmap).

Traffic Accidents in Porto Alegre, Brazil with Classification (marked map)

A marked map was used in this visualization to display a marker for each traffic accident that happened during a month in 2013. This visualization classified data based on the number of deaths. Here red markers represent traffic accidents with deaths and orange markers accidents without deaths. Figure 4.12 presents the generated visualization. Listing 4.5 describes the code for creating this visualization using GMaVis.

```
visualization: markedmap;
settings {
  latitude: field 42;
  longitude: field 41;
  page-title: "Acidentes Poa 2014";
  size: full;
  zoom-level: 11;
}
data {
  h-file: "/home/cleversonledur/Downloads/acidentes-2013.csv";
  structure {
    delimiter: ';';
    end-register: newline;
    date-format: "YYYYMMDD";
  }

  filter: field 9 is greater than date "20131201" and field 9 is less than date "20131230";

  classification{
    class ("Fatal"): field 13 is greater than 0 or field 14 is greater than 0;
    class ("N-Fatal"): field 12 is greater than 0 or field 11 is greater than 0;
  }
}
```

Listing 4.5 – Traffic accidents in Porto Alegre, Brazil with classification (marked map) code.

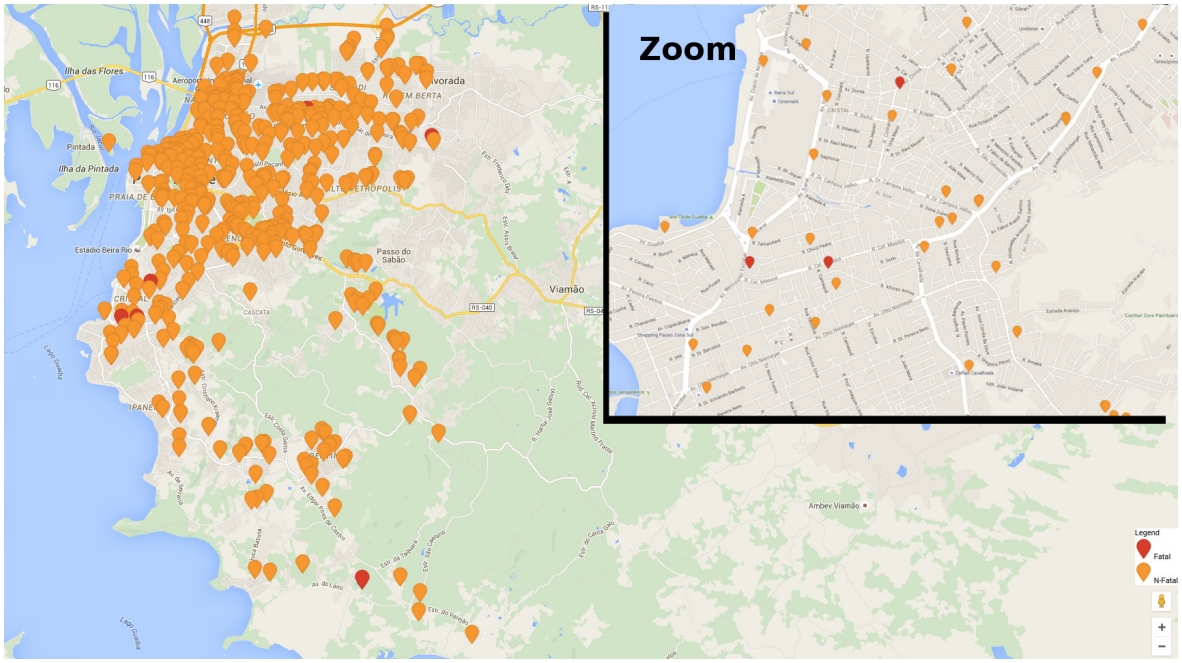

Figure 4.12 – Traffic accidents in Porto Alegre, Brazil (classified marked map).

Flickr Photos Took in 2014 Classified by Camera Brand Used

This visualization shows photo taken and uploaded to Flickr during 2014. The dataset used in this visualization has a field with information about the device used to take the photo. Then a classification was performed to display different color markers related to the camera brand. Figure 4.13 presents the visualization generated. Listing 4.6 describes the code for creating this visualization using GMaVis.

```
visualization: clusteredmap;
settings {
  latitude: field 12;
  longitude: field 11;
  marker-text: "Camera:" field 6 "<br><img src=" field 15 " width=200>";
  page-title: "Photos of 2014 Classified by Camera Brand";
  size: full;
}
data {
  file: "/home/cleversonledur/Documents/BIGDATA_YAHOO/yfcc100m_dataset-0";
  structure {
    delimiter: tab;
    end-register: newline;
    date-format: "YYYY-MM-DD";
  }
  filter:  field 4 is between date "2014-01-01" and date "2014-02-01";
  classification {
    class ("Canon"): field 6 contains "Canon";
    class ("Sony"): field 6 contains "Sony";
    class ("Nikon"): field 6 contains "Nikon";
    class ("Panasonic"): field 6 contains "Panasonic";
    class ("teste"): field 6 contains "Apple";
    class ("FUJI"): field 6 contains "FUJI";
  }
}
```

Listing 4.6 – Flickr photos taken in 2014 classified by used camera brand code.

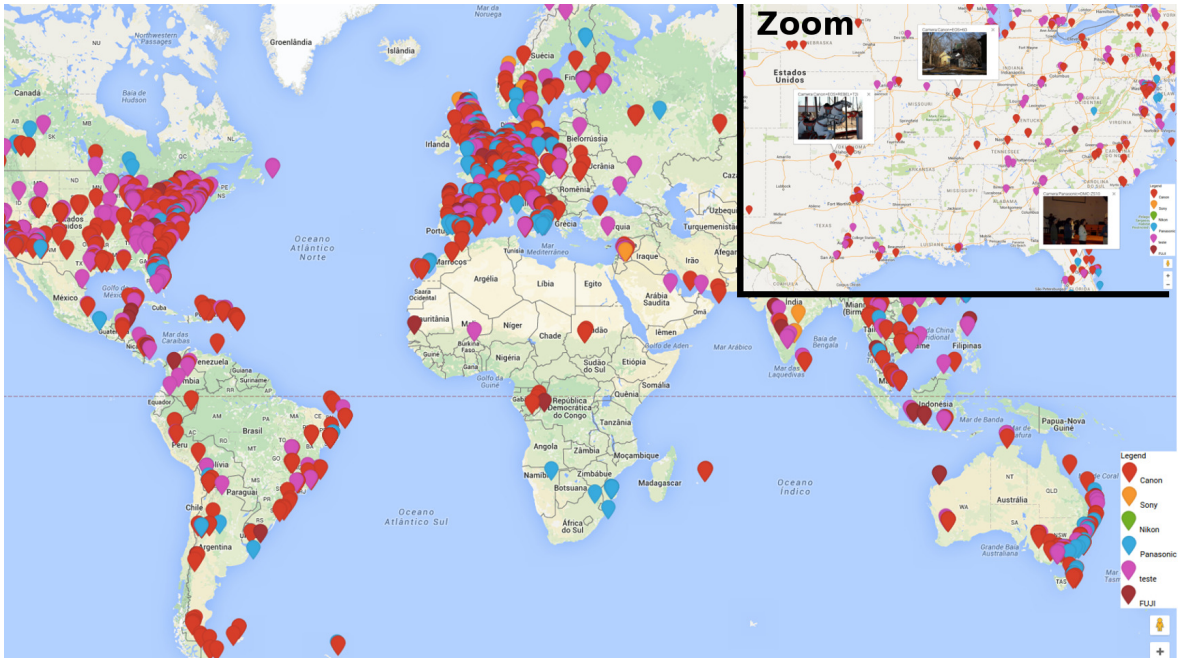

Figure 4.13 – Flickr photos taken in 2014 classified by used camera brand.