Demonstration

Visualization through Inductive Aggregation

Parke Godfrey * Jarek Gryz * Piotr Lasek *† Nasim Razavi *

York University, Canada *
Rzeszów University, Poland †
{godfrey, jarek, plasek, nasim}@cse.yorku.ca

1. INTRODUCTION

Visualization provides a powerful means for data analysis. To be useful, visual analytics tools must support smooth and flexible use of visualizations at a fast rate. This becomes increasingly onerous with the ever-increasing size of real-world datasets. First, large databases make interaction more difficult as a query across the entire data can be very slow. Second, any attempt to show all data points will overload the visualization, resulting in chaos that will only confuse the user.

Many solutions have been proposed to solve these problems,[1] but only one [1] addresses both of them simultaneously: *hierarchical aggregation*. Since it is not feasible to show all answers to a query, a natural way to reduce the size of the answer set is to aggregate it. We also need to support real-time interactivity; that is, to support an efficient way to move between levels of aggregation. Thus, we need a hierarchy of aggregations.

Hierarchical aggregation is not a new idea, of course. For data, it has been explored in OLAP, starting with the *data-cube* model. For images, it is only recent that new visual aggregation strategies have been developed for standard visualization techniques [1]. These strategies turn existing visualizations into multi-resolution versions that can be rendered at any desired level of detail. The *visual aggregate* can convey various information about the underlying data, such as their average, minima and maxima, and distribution.

Thus, data visualization systems face two challenges. First is an issue of *efficiency*. Most visualizations today are produced by first retrieving data from a database, and then using a specialized tool to render it. This decoupled approach results in significant duplication of functionality, while missing opportunities for cross-layer optimizations [5]. Second is an issue of *expressiveness*. Data visualization systems have not exploited modern graphics processing and rendering, due to architectural limitations, and lack of awareness. These graphics *shaders* meanwhile can significantly improve visualization.

We combine data aggregation with visual aggregation in a tightly coupled system that provides for smooth user interaction. Our implementation is based on a hierarchical data structure we call an *aggregate pyramid*.[2] By interacting with the pyramid in the back-end (via the database system), the front-end visualization client can quickly filter the data to move up and down in the aggregation hierarchy. We want the visualization mantra, "overview first, zoom and filter, then details on demand," [4] to be more like *skydiving* than gliding.

SKYDIVE's general architecture enables us to exploit modern graphics processing and rendering in new ways that other systems have not been able to exploit. Thus, for interactive data visualization, SKYDIVE is innovative in both *expressiveness*, by flexibly enabling new rendering techniques, and *efficiency*, due to tight-coupling in its architecture.

2. EXPRESSIVENESS

In SKYDIVE, a *data visualization* is defined by the user in two parts:

1. the *aggregate-pyramid query*, which defines the *dataset* cut from the database the user wishes to explore; and
2. the *visual mapping*, which maps the aggregate measures of the aggregate pyramid to visual *channels* in the *data texture* the user will explore.

In the next section, we define the concept of the aggregate pyramid in more detail, and how it is used to support *efficient* data visualization and exploration. For now in overview, we consider how it supports *expressive* visualizations.

Anatomy of the aggregate pyramid. The aggregate pyramid represents a hierarchy of aggregation levels we call *strata*. (This is visualized in Fig. 1.) The *base* of the pyramid represents the stratum with the highest resolution of our data (the data aggregated the least); higher strata represent successively lower resolutions (the data further aggregated). As with a *data cube*, the columns of an aggregate pyramid consist of *dimensions* and *aggregates*. Each tuple in the pyramid, called a *cell*, represents the aggregates of the raw data within the cell's area. The cells of any given stratum *tile* the dataset at the stratum's resolution. We consider here two-dimensional pyramids, with "X" and "Y" dimension columns.[3]

[1] See [2] for an overview.

[2] This concept is described in more detail in [3].

[3] One-dimensional pyramids are also useful for visualization, but with specific presentation models that we do not discuss here. Pyramids generalize to more than two dimensions,

The aggregate columns are defined over the *measures* of the raw data. For the pyramid, an *inductive-aggregate function* is defined in two parts, a *base* and an *inductive function*. The base function aggregates over the raw data to produce the cells of the *base* stratum of the pyramid. The inductive function then aggregates over the appropriate cells of the stratum below the current to compute the aggregate values for the cells of each stratum.

For example, consider scatter-plot data of event points—say fires that have occurred in the Seattle area—that we want to visualize. One aggregate we likely will want is to count events within each cell's area. The base function can simply be the aggregate count over the group-by into cells for the pyramid's base stratum. The inductive function is then sum, to sum up the counts of the sub-cells to compute the count for the cell.

It is important that the inductive functions are only allowed to look one stratum below, for efficiency of computation. This also means a good deal of care and thought must go into defining appropriate, meaningful inductive-aggregate functions that are effective for visualization.

Even with the simple example of scatter-plot data of events that have no specific qualities—an event simply occurred at a location—there are still a number of aggregates of this information in which one might be interested. *Count-per-area* is an obvious one, as discussed above. Additionally, there are statistical aggregates that can tell us something about how the events are *distributed* in the area represented by the cell. Are they uniformly distributed across the area, or are they highly clustered in given spots? An *entropy* function can be devised that offers a measure of such *dispersion* across the area.

If the scatter-plot data is richer, then there are many more aggregates we might wish to convey. Fire events might additionally carry a measure of *intensity*. We then may wish to convey information about the intensity of events in addition to the count of events. *Maximum* may be a reasonable aggregate over intensity, to convey the highest intensity of event to have occurred in an area (the cell). Another aggregate could carry the *standard deviation* over intensity of events within the cell. Fire events will also have *time* associated with them, and so forth.

The visualization mapping. The second thing that must be specified for a visualization is a *mapping* of the aggregates of the pyramid into visual *channels*. If the presentation model is a 2D image, these channels are the usual suspects from image processing: e.g., *red*, *green*, and *blue* (RGB) or *hue*, *saturation*, and *lightness* (HSL), depending on how one wants to consider the color space.

The mapping consists of functions that *map* (and *normalize*) the aggregates of the pyramid to available channels in the presentation model. The mapping should be done with care to be "orthogonal", so that each aggregate as mapped can be clearly distinguished. We envision developing a library of standard mappings to be available. SKYDIVE's architecture, however, is a general platform that allows for devising new, novel mappings.

The problem of channel paucity. A critical problem is the absolute paucity of channels for visual conveyance. If we are mapping to an image, we effectively have but three channels we can use in the mapping (e.g., HSL).

albeit presentation models for these are limited.

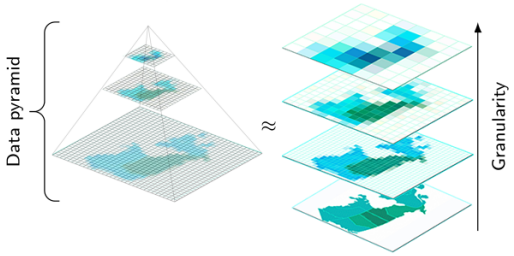


Figure 1: The Aggregate Pyramid Model.

While much work in data visualization has strived to address the problem of channel paucity—for instance, by graphical symbols, layout, and such to add effectively "channels" for conveyance—these do not work for interactive visualization in an inductive way, where one can zoom to change dynamically the degree of aggregation. Meanwhile, no work yet has taken advantage of the additional channels that the modern graphics environment afford us. SKYDIVE is designed to exploit just that, to great advantage.

Presentation models. The presentation model defines the "structure" that will be visualized. The model provides a set of channels that can be used by the mapping.

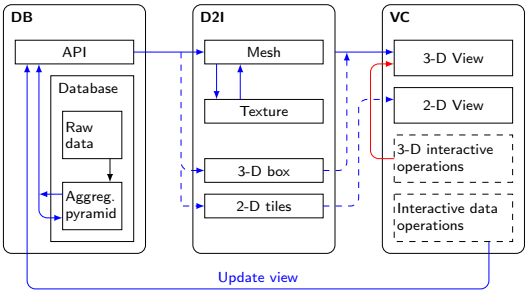
The 2D model. One model is that of a 2D image. The viewer manages the visualization as an image—which we call a *data texture*—within a canvas, allowing the user to zoom and pan around it. This view is, in essence, a *heat-map*. SKYDIVE's benefit is that the data texture makes it possible to explore dynamically this heat-map view progressively in realtime. This model suffers still from channel paucity, however; it is effective only if one can live within such a constrained channel space.

The 2½D model. A second model we call 2½D. For this, the model is rendered in 3D. The visualization now consists of two parts: the data texture, as before; and a *terrain*—a *manifold*[4] rendered as a *mesh*—onto which the texture is overlaid (*UV-mapped*).[5] This exploits modern 3D graphics rendering, which supports meshes and UV-mapping. This offers SKYDIVE additional channels of conveyance over points in the terrain: *elevation* (Z); *specular*; and *normal*. A specular map determines how "reflective" a point is on the surface. As scenes in 3D have external lighting, this is quite noticeable. A normal map dictates deviations of the normals, the "perpendicular" of a point with respect to the surface. By perturbing the normals of a neighborhood, that part of the surface can be made to look rough; leaving them as dictated by the mesh, the surface looks smooth. These are standard in graphics processing and used in game production for making scenes look more realistic. That is, these channels visually stand out.

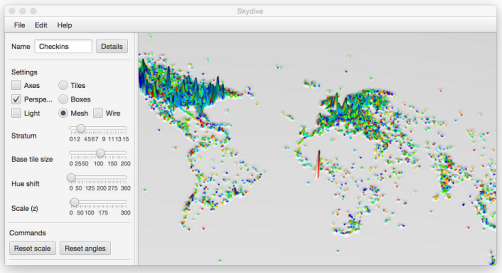
We can use the *alpha* channel additionally, as the terrain can be floated over a flat reference plane; the bleed-through of the reference through *translucency* of the terrain is readily

[4] A *manifold* is function that maps 2D coordinates to values. This can be rendered in 3D using *elevation*, "Z", over the 2D plane to indicate the 2D points' values.

[5] 2D images are called *textures* in this context, and the mapping of textures onto the mesh surfaces is called the *UV-mapping*.



(a) SKYDIVE's architecture.



(b) SKYDIVE's main window.

Figure 2: The SKYDIVE System.

obvious. This means we have effectively *seven* channels of conveyance in the $2^1/_2$D model, versus just the three in the 2D model.

Mixed models. SKYDIVE can mix presentation models for the same pyramid to provide simultaneous, synchronized alternative viewports into the same data. We also intend to support "cross-product" pyramids that could, for example, let one zoom and pan on XY and on T (*time*) independently.

3. EFFICIENCY

Structure of the aggregate pyramid. The idea behind this mirrors approaches taken by *progressive image formats* such as JPEG-2000. The image "pyramid" is multiresolution data structure that represents a $2^d \times 2^d$ image as a sequence of copies (2D arrays) of the original image, each "half" the resolution—half on the rows and half on the columns—of the next. Thus the base stratum of the pyramid is the full resolution version of the image, while the top stratum is a single pixel approximation of it.

Similarly, the *aggregate pyramid* represents $2^d \times 2^d$ data cells at its base, with each subsequent stratum halving the "resolution" (doubling the aggregation). Construction of a aggregate pyramid can be accomplished efficiently by the database engine by building it from the base upwards. First, the base stratum is created by aggregating the raw data into the base cells. Then subsequent strata can be produced recursively by aggregating spatially the constituent quadrant cells of the stratum below. The cells can be indexed by stratum, and by Hilbert order that linearizes their order, which then can be indexed via a B+-tree. As such ordering preserves locality of sub-cells that are needed to merge for the next higher stratum, a stratum can be produced in proper order by a single scan of the cells of the stratum below it.

SKYDIVE employs the aggregate pyramid to preprocess data so the visualization process can be handled efficiently. Given a query defining the dataset to explore, the database system *materializes* the aggregate-pyramid version of the dataset query, and indexes the pyramid by stratum and Hilbert order, as discussed above.

The materialized pyramid is managed by the database engine during the visual exploration of that dataset. Interactive operations at the visualization client are then supported by querying into the aggregate pyramid at the appropriate stratum and range (bounding box), which can be handled efficiently and at real-time, interactive speeds. Thus, SKYDIVE tightly couples database support for processing the data with the interactive visualization.

Operations. SKYDIVE is designed to support the following visual operations over the dataset for visualizing. Each, in turn, can be supported efficiently by the database system over the materialized aggregate pyramid.

Resizing. The user can change the current viewport by changing the size of the visualized data (up or down). In sizing, the visualizer may need to present a different level of resolution. For instance, in a size-up operation, the user requests a higher resolution image. As a result, the system needs to retrieve the aggregated data from a higher resolution stratum in the pyramid.

Zooming. The user can request to view more detail of a part of the image by specifying a *window of interest*, selecting a portion of the image by zooming in. The system maps the requested window to the stratum with high enough resolution to fit the canvas, and selects the appropriate range.

Panning. In panning, the user changes the viewport in the image, but within the same level of resolution. If the user pans, the system will check the availability of the visual data in the current stratum, and request the additional range from the pyramid in that stratum.

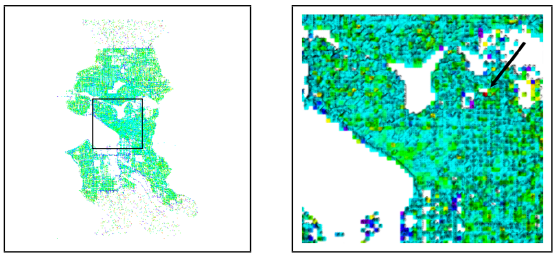
4. ARCHITECTURE

Skydive's components. SKYDIVE is composed of three main components, as shown in Figure 2:

- the Database Module (DB);
- Data-to-Image module (D2I); and
- the Visualization Client (VC).

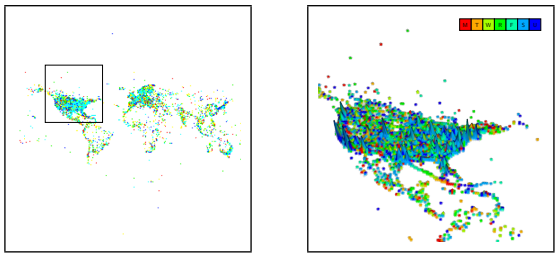
Each is designed to use a different type of *computer memory*. The DB module uses *disk* to store and manage the raw data, and materialized aggregate pyramids. The D2I module works with a small subset of the aggregated dataset, and stores data in *main memory* (RAM). The VC module uses the *graphic card's* capabilities to perform more advanced operations—such as zooming, scaling, panning, and rotation—over the graphical representation of the data.

This separation of concerns provides useful flexibility. Each component can be implemented as a separate service, deployed on a different machine. This leverages the idea of *compression* of data conveyed between the modules, letting us implement a tightly coupled visualization system. The SKYDIVE prototype is implemented as a desktop application with the three modules as described above and shown in Fig. 2a. The main window of SKYDIVE, shown in Fig. 2b, is



(a) Overview image. (b) Zoomed in, terrain view.

Figure 3: Visualizations of the Seattle 911 Dataset.



(a) Data texture of check-ins. (b) Zoomed in, terrain view.

Figure 4: Visualizations of the Brightkite dataset.

composed of a few simple elements: an upper menu for performing basic file operation; a left panel for tuning a loaded and currently displayed visualization; and a visualization view for rotating, panning and zooming.

Graphical variables. The user also defines the presentation model to be used and the visualization mapping (of aggregates to channels) to be employed, as discussed in §2. The system prototype supports three presentation models:

- a 2D heat-map;
- a $2^1/_2$D heat-map by 3D barchart; and
- a $2^1/_2$D terrain (by mesh and UV-mapping).

Generating meshes and textures. The data texture is generated by the D2I module by selecting the appropriate window out of the aggregate pyramid, and applying the visualization mapping. For the $2^1/_2$D terrain model, a mesh is additionally computed by the D2I. The mesh is created based on a stratum of lower resolution, for better visual appeal, and for efficiency in the VC.

User interface. The user interface, as shown in Fig. 2b, allows the interactive visualization operations, as discussed above. The user can scale, translate, and rotate the currently displayed visualization in the Visualization Client (VC). The VC is implemented using JavaFX, which natively supports these functions. Scaling, translation, and rotation do not require to query the aggregate pyramid, hence are performed entirely within the VC, supported by the GPU.

Other *interactive* functions do require queries to be issued from VC to the DB module. For instance, if the user wants to focus more on a certain area of a visualization, then the system must request the data from the appropriate stratum; of higher resolution for zooming in, and lower for zooming out. Issuing such a request results in the loading and generating of the mesh and texture by the D2I. The VC then displays this using the GPU's graphics pipeline.

5. DEMONSTRATION SCENARIO

Datasets. We test SKYDIVE using several datasets, two of which are described below.

9-1-1 calls. The Seattle Police Department 911 Incident Response dataset [6] contains over one million records. Each represents the police response to a 911 call within the city. Fig. 3a shows a *density* map of the calls. Color denotes the number of calls made within the area represented by a pixel. Based on the plotted heat-map, a user is not able to conclude anything more than that there are some areas of slightly higher density than their neighbors.

In Fig. 3b, we have switched to the terrain view, where elevation indicates the density, and color refers to a most frequent type of a call within the cell. In this view, we see more detail. For example, the red pixel indicated by the arrow represents unusual activity within the magnified area.

Brightkite check-ins. The Brightkite Check-ins Dataset[7] consists of over four millions records of geographical positions reported by users of a geo-location social service. In Fig. 4a, the heat-map represents the dataset over one measure: color represents days of week for which user activity was highest within the areas represented by the pixels. In Fig. 4b, a terrain map is shown of a zoomed in portion with more in the mapping. The texture color again denotes day of week with highest activity. Elevation denotes number of check-ins. We can deduce that weekends were most active days for Brightkite users in the USA. We can additionally see the areas in which the most users were active.

Richer mappings. We will demonstrate the richer mappings offered by the $2^1/_2$D model with normal, specular, and alpha channels. These are not easy to show in static pictures, but stand out in display in the demo. With these, additional aggregates can be conveyed to a viewer simultaneously. Roughness of the surface (a normal map) can be used to represent variance of a measure within cells. Shininess (a specular map) can be used to show spatial dispersion within the area represented by a point on the terrain.

6. REFERENCES

[1] N. Elmqvist and J. Fekete. Hierarchical aggregation for information visualization: Overview, techniques, and design guidelines. *IEEE Trans. Vis. Comput. Graph.*, 16(3):439–454, 2010.

[2] P. Godfrey, J. Gryz, and P. Lasek. Interactive visualization of large data sets. Technical Report EECS-2015-03, York University, March 2015.

[3] P. Godfrey, J. Gryz, P. Lasek, and N. Razvi. Skydive: An interactive data visualization engine. In *IEEE Symposium on Large Data Analytics and Visualization, Chicago, USA, October 25-26.*, 2015.

[4] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Visual Languages, 1996. Proceedings., IEEE Symposium on*, pages 336–343. IEEE, 1996.

[5] E. Wu, L. Battle, and S. R. Madden. The case for data visualization management systems [vision paper]. *Proceedings of the VLDB Endowment*, 2014.

[6] https://data.seattle.gov/

[7] https://snap.stanford.edu/data/