HIERARCHICAL AGGREGATE STRUCTURE BY INDUCTIVE AGGREGATION FOR INTERACTIVE DATA VISUALIZATION

NASIM RAZAVI

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

MASTER OF SCIENCE

GRADUATE PROGRAM IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
YORK UNIVERSITY
TORONTO, ONTARIO

SEPTEMBER 2016

Abstract

Data visualization is a method to facilitate the process of knowledge discovery and decision making. Effective and practical visual analytic systems have to support real-time and smooth interaction. Traditional data processing techniques and systems are inadequate when it comes to large datasets. The powerful features of relational database engines can be adapted to facilitate the visual representation of very large datasets.

In this thesis, we propose techniques to provide a tightly coupled system with a database engine back-end to support a data visualization front-end. We employ data reduction techniques along with other methods to form a hierarchical data structure that we call the *inductive-aggregate* pyramid to provide multiple representations of data. We also propose a generalized form of inductive-aggregate pyramids that we call *cubed pyramids* to provide a richer representation of high-dimensional data. We develop and employ techniques to build and query efficiently these structures to support interactive data visualization.

Acknowledgements

First, I would like to express my sincere gratitude and appreciation to my supervisory committee Professor Parke Godfrey and Professor Jarek Gryz for their continuous support of my research with their patience, motivation, enthusiasm, and immense knowledge. I am grateful for the guidance and great effort they put into training me in this scientific field. Their keen insight and constant encouragement helped me learn to believe in myself.

I wish to sincerely thank my thesis advisory committee member Professor Henry M. Kim for his time and valuable suggestions and comments to improve the quality of my thesis.

I would like to express my very sincere gratitude to Professor Piotr Lasek from University of Rzeszow for his support and guidance to make this thesis possible and help me in this project by his deep knowledge and advice and giving me a better understanding of the problem.

I am thankful to Professor Nikolay Yakovets from Eindhoven University of Technology for his constant support, encouragement, and his technical assistance. He has helped me in this journey by generously sharing his professional experience and providing the valuable guidance.

I want to thank York University, the Faculty of Graduate Studies, and Lassonde School of Engineering for their support.

I would like to take this opportunity to express my profound gratitude from the deepen of my heart to my beloved parents, for their love and encouragement.

Finally, I would like to express my very sincere thanks to my beloved husband for his love, patience, and continuous support that led this journey to a success.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 The Problem: Seeing Data

Managing, analyzing, and processing data in today's information overloaded world is a significant problem. Datasets such as scientific data are growing rapidly. Visualization is a powerful means to convey knowledge hidden in datasets in order to gain deeper understanding of data. Data visualization can lead to good analysis and decision making, especially over large datasets. Visual representation of data allows for rapid and easy exploration, leading to comprehension of the underlying information by revealing connections, relations, patterns, and anomalies. Furthermore, visual exploration of data helps to achieve insight about what was unknown.

Card et al. [1], in their paper where an organization of the information visualization literature is presented, define visualization as a means to map data into a visual vocabulary that provides interactive data exploration and analysis. Stephen Few [2] defines data visualization as graphical representation of abstract information that maintains two principles: communication and data analysis (sense-making). MacEachren et al. [3] also illustrate these principles as communication and insight discovery. In their work where they propose a cartographic visualization model, they state two important roles of cartographic visualization in data exploration and analysis. They make a distinction between a cartographic communication model (communication tools) and a cartographic visualization model (visualization tools). A cartographic communication model can be developed to depict known patterns and relationships to simplify communication. A cartographic visualization model, on the other hand, unveils hidden knowledge and insight into the data which was previously unknown. Data visualization not only represents the known aspect of data such as relationships and connections to facilitate communication, but also identifies new patterns providing insight into the data.

The datasets that organizations create are ever larger nowadays, commonly continuing millions to billions of elements. An individual data element, say, a tuple, will often contain numerous attributes. Data visualization maintains a graphical representation of abstract information, concepts, and relationships by characterizing and categorizing spatial and temporal attributes. Many datasets contain data of high dimensionality and complexity. Therefore, traditional visualization approaches are not efficient nor sufficient to provide visual representation of very large datasets of high dimensionality along with variety of attributes. To have an effective and expressive visual representation of high-dimensional and complex data, understanding the characteristics of human visual perception is vital. Alexandre et al. [4] present how to implement more efficient visualization systems by taking important aspects of human visual perception into consideration. They classify visualization activities in three categories: exploratory analysis; confirmatory analysis, and presentation. The purpose of exploratory analysis is to convey new knowledge by revealing insight, visual representation of relations, and hidden patterns, which can lead to creating hypotheses. In confirmatory analysis, the graphical representation of data is explored to confirm or disprove a predefined hypothesis. In presentation, graphical representation of data is used to expose relations, characteristics, patterns, and anomalies.

Many techniques have been proposed to map high-dimensional data with large numbers of attributes into graphical metaphors which are coherent and meaningful in an effective and efficient way [5, 6]. Yet, visualizing very large datasets with millions or billions tuples remains a great challenge. Scaling large datasets to any desired resolution considering the limited number of pixels on displays in an efficient manner, has recently gained much attention in data exploration and visualization research work. In this thesis, we study the issues and challenges in integrating database systems and visualization techniques to facilitate interactive exploration and visualization of large datasets. We propose structures and techniques to construct a multi-resolution dataset of various levels of detail. We address the challenges involved to construct a multi-resolution dataset efficiently that can be used at run-time to support real-time, interactive visual exploration of data.

1.2 Motivation

Data visualization and analysis systems can provide effective and powerful means to reveal trends, unusual patterns, and anomalies in data. Gray et al. [7] illustrate four steps in data analysis: formulating; extracting; visualizing; and analyzing. Formulating defines a query needed to retrieve data from a dataset. Next, aggregate data needs to be extracted. The processed and computed result is then ready to map into a graphical representation, which is visualization. The final visual result is then analyzed to form new hypotheses, then to query the data in follow up as needed.

A challenge in data visualization is how to map data into graphical metaphors that provide an exploratory representation of data which leads to patterns and anomalies discovery, especially for high-dimensional datasets. Nowadays, another major challenge is the extreme amount of data that needs to be processed and visualized. Datasets can be enormous, while the display screen has limited number of pixels; and human cognition can only assimilate but so much at any given moment. Thus, an important question is how we can sufficiently visualize data that summarizes underlying datasets when they are very large. This requires a way to reduce the size of the original dataset, to summarize it, to make the visual representation of data possible with a limited number of pixels. There are many challenges involved. One must determine how to efficiently reduce the size of data while assuring that the reduced version still tells the same story.

A pertinent problem is how to reduce data in a precise and coherent manner that scales to different screen sizes with different resolutions. Managing, storing, and manipulating large datasets are each expensive processes. Data visualization systems such as Tableau [8–10] have their own data management systems. DBMSs for many decades, have provided powerful and efficient data storage, management, and manipulation. These systems are well designed to store and process data optimally. Database systems can be adopted to facilitate the complex process of visual representation of data from very large datasets with millions or billions of tuples.

DBMSs can be integrated with data visualization systems to push the data processing into the database engine. A number of research efforts have been conducted to determine how relational engines can efficiently support data extraction and visualization systems [11]. Database techniques can be applied to reduce the load on the visualizer (visual space) by reducing the amount of data to be visualized. Standard features in SQL can be employed to deal with large dataset exploration. Data reduction techniques such as aggregation along with optimization tools embedded within database systems, are some of the features that can facilitate the process of data exploration and extraction. An approach is to reduce the size of very large datasets by using aggregation in the data space (database). Scalar [12], for instance, applies aggregation, sampling and filtering operations to reduce the size of the results along with analysis queries using optimization techniques to predict the size of the final result. Hierarchical aggregation is another approach to build a multi-scale representation of information in order to scale the visual representations [13].

In addition to the size of a dataset, high dimensionality of a dataset imposes more complexity on data extraction and visualization. Aaggregation across different dimensions is required when exploring data for hidden knowledge and patterns, anomalies, and decision making. In data visualization and analysis, dimensionality reduction is performed by summarizing data along different dimensions [7]. Gray et al. [7] introduce data cubes as database relations constructed by aggregating data across all combinations of dimensions. This is the generalization of the operator *group-by.* Data cubes provide for efficient exploration and analysis of high-dimensional data for decision making, knowledge discovery, and finding patterns and anomalies. However, since in data cubes, aggregation is performed for all possible combinations of dimensions, the size of a data cube exponentially grows with respect to the number of dimensions.

Despite the expense of constructing data cubes, these structures are prominent in data exploration and analysis. The OLAP extensions to the SQL standards provide operations such as roll-up and drill-down over data cubes. The roll-up operation creates hierarchies over dimensions by aggregating, with respect to a concept hierarchy of dimensions. Spatial aggregation, however, is not directly supported in OLAP in the SQL standards. Research efforts have been conducted to propose techniques and solutions to make the construction and storage of data cubes optimized and efficient. Cubes can be employed to maintain the multi-scale representation of high dimensional data [14]. ImMens [15] relies on a binned aggregation technique [16], along with partial data cubes combined with the roll-up operation, to form hierarchies over time. Nanocubes, proposed by Lins et al. [17], are reduced size cubes that are small enough to fit in the main memory to maintain real-time and interactive data visualization and exploration.

Besides the complexity and difficulty behind high-dimensional data visual representation and exploration, data needs to be scaled the size of the screen. Whenever the size of the given dataset is larger than the number of pixels, the dataset has to be processed to provide a result set compatible with the size of the screen. Shneiderman's mantra [18] can be followed: "Overview first, zoom and filter, then details on demand." When a dataset is too large to fit the screen, an *overview* can be displayed. The user can then zoom in to observe more detail as needed. Providing more detail requires performing new queries to retrieve data at "higher resolution". Every time the user sends such a request (indirectly, by manipulating the visualization interface), a query needs to be performed in the background. Querying directly the very large base dataset for every single such request, followed by processing and extracting of data to be visualized, is too expensive for real-time, interactive systems. Going back and forth between the front-end (visual space) and the back-end (data space) to extract and process data imposes a significant delay on the user and system interaction. Every time a request is sent, the user has to wait for the response to be processed.

A solution to have real-time interaction is to pre-compute data at different levels of detail. Storing data in a multi-resolution structure alleviates the expensive need to re-visit the dataset each time. With a hierarchical data structure with multiple levels of detail pre-computed, the appropriate resolution and "slice" to fit the request can be retrieved directly. Perrot et al. [19], for instance, propose an architecture to support interactive visualization and exploration of large scale data over a pre-computed multi-level aggregation of data. A hierarchical structure facilitates the process of exploration and extraction of data when multiple representations of data is desired. Choosing a proper index provides for efficient range selection over the multi-resolution structure. A pre-computed structure with various levels of detail supports real time interaction since the user's request does not have to be computed and processed over the original dataset.

1.3 Methodology

In this thesis, our objective is to define techniques and procedures to define and build efficiently a hierarchical structure with levels of detail that supports interactive data visualization and exploration of large datasets; especially when different data resolutions are desired. To achieve this goal, we borrow from concepts of techniques such as progressive image transformations. We adapt these techniques to provide data summarization in a hierarchical form which we call an inductive-aggregate pyramid. An aggregate pyramid defines the multi-resolution aggregation of the dataset that represents corresponding aggregate data to be visualized. We provide multi-resolution aggregates of the datasets by defining techniques to aggregate data spatially over desired dimensions. — The structure is then indexed on one-dimensional ordering values ($Z/Morton$-order [20]) and the level of detail. The proposed index efficiently maintains range queries over the hierarchical structure to support interactive operations such as zooming and panning. — A request is processed and data with the required size and resolution is extracted from this pre-computed hierarchical structure in an efficient way. The result is then sent to the front-end (visualizer) to be visualized.

Multi-scale representation is recently a prominent issue which has been addressed in the image processing and database communities. In image processing systems, large images need to be transmitted and displayed in systems with finite transmission bandwidth and limited screen sizes. In image representation and transmission, techniques have been proposed and employed to improve the performance, especially for systems with restricted resources [21]. Image compression is one of the techniques to reduce the usage of transmission bandwidth and storage. Image compression techniques also provide progressive representation of an image by gradually transmitting the image data.

JPEG2000 [22], as an image compression and multi-resolution format, maintains progressive mode for image compression and transmission. In progressive mode, an image is encoded into multiple copies. In the first scan, a very coarse and low quality version of the image is displayed. This low quality version is then refined by receiving subsequent scans with more detail. This process continues until a high quality image with desired resolution is generated. A wavelet compression technique employed in JPEG2000 maintains the progressive transmission. Encoded data is progressively sent and decoded to maintain a gradual refinement of an image. Progressive transmission improves interaction by providing a quick lower resolution of the image.

Platforms, nowadays, spam a variety of representations and different screen sizes. This requires multi-scale representation of images. Thus, representing and analyzing images at various scales is required. To process and analyze the initial large images in order to search and extract features, a low resolution copy of an image can be examined. To maintain the multi-scale representation of an image, a pyramidal structure can be employed. A pyramidal structure successively provides multiple levels of representation of an image. An image pyramid is a hierarchical structure containing different copies of an image with different resolutions arranged from high quality image resolution to a low quality copy. The low resolution levels can be efficiently computed through recursive algorithms. Tanimoto et al. [23] represent an image pyramid as a multi-resolution representation of an image which is a stack of arrays where the size of each array is one-quarter the size of its preceding computed array.

Hierarchical structures have been employed in many systems such as image processing, computer graphics, and geographic information systems. They have also obtained much attention in database systems specifically as index structures for spatial databases [24]. Samet [25] represents a class of hierarchical data structures called quadtrees which all comply the recursive decomposition basis. Hierarchical structures facilitate the process of narrowing down and extracting a desired subset of data in an efficient way [26].

Samet [26] defines an image pyramid of a given $2^n \times 2^n$ image as a sequence of image arrays I_n to I_0 where I_n represents the highest quality version of the image and I_0 represent a single pixel. The image array I_{i-1} can be constructed from I_i with half of the resolution in a recursive decomposition process by subdividing the high quality versions into low-resolution copies. The size of each version I_{i-1} ($2^{i-1} \times 2^{i-1}$) is one-quarter the size of its preceding image array I_i ($2^i \times 2^i$). Although Samet [26] represents quadtrees and pyramids as similar and related structures, he makes a distinction between quadtrees and pyramids in various aspects. For instance, in contrast to pyramids which are multi-resolution data structures, quadtrees are variable-resolution data structures. pyramids are also appropriate structures to focus on the limited scopes to search and extract features as opposed to quadtrees.

Implementation and use of these hierarchical structures are efficient. To improve the manipulation of hierarchical structures such as quadtrees, these structures can be stored in a linear order. The linear quadtree [27], for instance, is defined as a class of data structures where leaf nodes are represented by a set of sequential numeric keys. In linear quadtrees, a quadtree is addressed and stored more efficiently on the disk. Shaffer et al. [28] propose an algorithm to compute the linear quadtree. The algorithm assigns a sequence of key values to each pixel representing the leaves of the quadtree. The linear ordering employed in this approach is called Morton ordering, introduced by Morton to index maps in a geographic information system [20]. Morton order is a space filling curve to map a high-dimensional space to a one-dimensional set. The multi-dimensional ordering also known as space-filling curve, introduced by mathematician Giuseppe Peano [29], continuously maps multi-dimensional data points onto a one-dimensional domain by sequentially assigning a unique number to every point in the multi-dimensional space. The locality of points is preserved in the linear ordering.

To have an interactive visual representation and exploration of data in very large datasets, we propose a hierarchical structure called the *aggregate pyramid* to store and query data at different resolutions. In order to summarize and store data in a multi-resolution structure, we define techniques to aggregate data progressively starting from raw data of the original dataset. In this approach, data is processed and computed in advance to form a hierarchical structure at different levels of detail. To create the aggregate pyramid, we define a technique to aggregate data spatially over one or more dimensions called *inductive aggregation*. In this technique, the base of the pyramid has to be processed and materialized first. Inductive aggregation is then performed over bins of last previously computed level to form the next lower resolution level or stratum. Since, a stratum is computed from one immediate stratum below, in building a new aggregate level with less detail, we need to only process the last preceding aggregate level. Inductive aggregation is an iterative process that progressively creates aggregate levels (strata) from last computed stratum. In fact, inductive aggregation performs aggregate functions on a group of four adjacent bins or tuples to form a super bin of the immediate next higher stratum with lower resolution.

To build the aggregate pyramid progressively, we determine two phases: base-aggregation; and re-aggregation. In the base-aggregation phase, raw data from the given dataset is aggregated and processed to form the base of the aggregate pyramid. Each tuple in the base represents a bin. To make the process of constructing the aggregate pyramid more efficient, we apply a multi-dimensional ordering ($Z/Morton$ ordering [20]) on the raw dataset to map a multi-dimensional dataset into one-dimensional data points while their spatial locality is preserved. The processed and aggregated dataset is then sorted and written sequentially in Z ordering values onto the disk. Because of the Z-order, adjacent points are likely to be also close in this order. In the re-aggregation phase, the inductive aggregation is progressively applied on the base stratum to form the next higher stratum with less detail. In this phase, starting from the base, every four adjacent points are aggregated to form a new aggregate value representing a bin in the next higher stratum. A subsequent stratum is materialized from the previously built stratum, starting from the base until the highest stratum containing one bin is built. Because of the ordering technique, spatially adjacent points in the recently computed stratum are likely to be consecutive on the disk. To build the next level with less detail, we only sequentially read the tuples (bins) from the previous computed stratum once. The tuples in stratum i are sequentially processed in a single scan to produce the stratum $i-1$. Since tuples in stratum i are sorted on Z-order, the tuples needed to be aggregate are adjacent on the disk in the sequence. The stratum $i-1$ is also written onto the disk sequentially in Z-order.

A multi-dimensional mapping such as Z ordering (Morton order [20]) supports *B+-tree* indexes over high-dimensional datasets. We create an index on stratum and one-dimensional ordering (Z-order) to facilitate building and querying the aggregate pyramid. With an index on stratum and Z-order values along with taking advantage of quadrant-recursive property of Z ordering [30], the aggregate pyramid sufficiently maintains interactive operations such as re-sizing and panning. Since the aggregate pyramid is constructed in advance, the user's request does not require to be processed over the original dataset. The request is handled by querying a slice of the aggregate pyramid matching the desired size and resolution. If the requested size is too large to fit the screen, the same window of interest but from a lower resolution stratum is retrieved and visualized. We propose algorithms to query efficiently the aggregate pyramid, exploiting the quadrant-recursive property of Z-order.

To provide details by representing all individual data points in an aggregate pyramid, we need to determine the depth at which all points are separated. We call this dept, the natural depth. To construct the aggregate pyramid, this natural depth of the pyramid has to be determined. The deeper the structure is, more detail it can provide; but one has to consider limiting depth, since a very deep stratum might be sparse. Depth has a direct impact on the performance of the aggregate pyramid construction. When a stratum is sparse, aggregation to the next stratum does not result in a reduction of data. Starting at too shallow, on the contrary, misses detail and the pyramid might not provide sufficient detail and desired resolution. We address this tradeoff via experiments over different datasets.

To represent multi-resolution of high-dimensional datasets, we propose cubed pyramids. Cubed pyramids are cross products of aggregate pyramids. An aggregate pyramid can be either a one-dimensional or multi-dimensional dataset. In a cubed pyramid, inductive aggregation can be applied over a dimension or groups of dimensions independently. We call a single aggregate pyramid processed in building the cubed pyramid an axis. Each axis can be constructed by applying inductive aggregation over one or more dimensions. Cubed pyramids can be defined as a generalization of roll-up data cubes. In contrast to data cubes [7], cubed pyramids are built by aggregating data over desired and predefined dimensions, not all combinations of dimensions. They also provide for exploration over each axis independently. For example, providing the visualization of a temperature map, the user may ask to view the temperature data of the map at different times (time intervals). The user can view the temperature map of a day, a week, or a year without the necessity of sending more requests to the database engine and waiting for a response.

1.4 Contributions

The following list includes the summary of our contributions:

1. Provide a comprehensive survey of state of the art of database support for interactive visualization. (Chapter 1 and Chapter 2)
2. Propose a data structure to store data in multiple levels of detail called inductive aggregate pyramid. (Chapter 3 and Chapter 4)
   - (a) Define techniques to inductively construct the aggregate pyramid. (inductive aggregation)
   - (b) Define the meaningful depth for the aggregate pyramid
   - (c) Design API to support interactive visual operations

3. Implement efficiently the aggregate pyramid materialization and use of inductive aggregation.

    (a) materializing efficiently the aggregate pyramid

    (b) indexing the aggregate pyramid for fast API evaluation

    (c) Determining efficiently the natural depth and representing its critical impact on the efficiency of building the aggregate pyramid

    (d) Evaluating API cost in an interactive speed

4. Generalize aggregate pyramids to cubed pyramids (Chapter 5)

    (a) Define multiple axes for richer representation of high-dimensional data

    (b) Implement efficiently materialization and use of cubed pyramids.

    (c) Demonstrate how cubed pyramids' richer presentation supports for interactive data representation

5. Verify experimentally efficiency of aggregate pyramids (Chapter 6)

    (a) Estimate the aggregate pyramid cardinality in advance

    (b) Show the fact of natural depth on cost

1.5 Outline

The thesis is organized as follows. In Chapter 2, we present a summary of related work regarding the integration of database systems and data visualization and exploration to provide an interactive visual environment. In Chapter 3, we propose, the inductive aggregate pyramid, a hierarchical data structure to represent data at different levels of detail. We develop techniques and algorithms to build and index the aggregate pyramid. In Chapter 4, we illustrate how aggregate pyramids support interactive operations for data visualization and exploration. In Chapter 5, We develop the concept of cubed pyramids, the cross product of aggregate data pyramids. In Chapter 6, we present experiments conducted on two different datasets to determine building and using aggregate pyramids. In this chapter, the importance of the natural depth and its impact on the performance are experimentally determined. In Chapter 7, we conclude with a summary and a discussion of future work.

Chapter 2

Background and Related Work

Processing, analyzing, and exploring data effectively and efficiently have been great challenges in data visualization. As data has grown rapidly in recent decades, this has become even more challenging in processing time due to the size of the data to be processed. With data produced and stored even more quickly, an important challenge in data exploration and visualization is scaling visual presentation from thousands to millions or even billions of data items (records). The integration of database and visualization systems can make database analysis tools accessible for visualization systems in data exploration and presentation. Numerous work has sought to overcome the difficulties of data exploration and analysis combined with visualization techniques.

Since database systems are designed to host data to be stored, manipulated, and queried in recent decades, they are natural tools for data exploration and visualization. They provide powerful means to explore data with their search engines in an effective way. One of the earliest attempts in data exploration and relational information visualization is APT (A Presentation Tool) [31]. APT is a pioneering work in this area. It was developed to provide presentation tools and graphical techniques for automatically designing two dimensional presentations of relational information. Mackinlay [31] describes graphical presentations as sentences in a graphical language. Using graphical presentations raises the issue of how graphical designs can present a wide variety of information in an expressive and effective manner. Specifically, he defines the key graphical design issues as expressiveness and effectiveness. Expressiveness determines whether a graphical design can express the relevant information correctly. Effectiveness determines whether a graphical presentation provides a meaningful and understandable graphical output compatible with the human visual system. Thus, the idea behind this work is designing and developing a presentation environment where information extracted from a database can be presented in expressive and effective graphical forms. Unlike other work, the key point of APT [31] was producing a wide range of comprehensive graphical designs for relational information. SAGE [32], a research effort in automating the presentation of relational information, investigates the needs and problems of developing tools to simplify the designing and programming of displays which support data presentation in large information systems. SAGE can be said to be the improved version of APT [31] with the additional support of interactively creating and designing complex combination of data presentation displays in order to meet the application requirements. It also provides the possibility of visualizing the intermediate results to help system developers discover outliers and anomalies. The intermediate results can also assist system designers and developers to understand behavior of systems.

Many efforts have been done to provide more interactive data visualization and flexible graphical user interfaces to explore relational information at the time that thousands, or even hundreds, of data items (tuples) were considered as large data sets. VIS [33] and IVEE [34], for instance, rely on some principles such as dynamic query filters, starfield displays [35], and tight coupling to create an interactive visual environment for relational information exploration. Dynamic Queries (DQ) [36] is a concept of direct manipulation of databases by graphically formulating and representing the queries and their result sets with graphical widgets such as sliders. It provides quick feedback of the results every time a user adjusts the query parameters. Moreover, because of its graphical environment, this interface is easy to learn and work even for non-expert users. Starfield displays are two-dimensional scatter plots for visual representation of result sets which provide some additional operations such as zooming and panning. One of the important aspects of tight coupling is the progressive refinement of queries in which the user can modify the query parameters when the query result is large in order to eliminate the unwanted data and have more restrictive results. It means that the output of a query can be processed and explored as the input for a reformulated query.

Another aspect of tight coupling applied in VIS and IVEE is detail-on-demand selection. At the time, one of initiate approaches to conquer the issue of dealing with large data sets was to visualize as many data as possible by using each pixel of the display, each to represent one data item of the result set. Therefore, each record in a result set is represented as a point for saving space in order to display more data points. The user then can select any data point to see more detail (desired attributes of that data point). Another interactive visualization and query system implemented based on the concept of dynamic queries is Spotfire [37]. Spotfire provides a high level of database exploration and visual information representation by attaching graphical objects to database objects. Even with these efforts done to make data exploration and visualization more effective at the time, the problem of visualizing tens of thousands data points remained a great challenge. The issues of the limited number of objects that can be visualized in the display and overlapping points in the scatter plots, were still needed to be addressed.

In the 1990's, another step forward in interactive data visualization with database integration was taken by IDES [38]. IDES is a knowledge-based interactive data exploration framework integrating data exploration and automatic presentation systems to discover hidden relationships in large data sets from extracted data. Not only the automated presentation of large data sets is considered as a big challenge, but also the necessity of an interactive data manipulation and analysis is emphasized. Interactive data visualization is an iterative process of data exploration by its nature. The authors [38] divide the process of data exploration and visualization into three sub-processes: data visualization; data manipulation; and data analysis. The data visualization operation is the process of designing and creating a view using effective graphical presentations. The data exploration and visualization are dynamic processes that allow the user to change the visualized data on the display in order to see the required information. IDES identifies three categories of data manipulation operations: controlling the scope; selecting the focus of attention; and choosing the level of detail. In IDES, users can choose what amount of data to be visualized (controlling the scope), what attributes they want to be presented (selecting the focus of attention), and finally what granularity of data they want to examine (choosing the level of detail) using aggregation or decomposition. It can be seen clearly that the discordance of the screen size and the amount of data in large data sets makes the idea of data reduction and visualizing data at different levels of detail inevitable.

While visualization makes data exploration and analysis more effective and comprehensive, a good design and development of a data visualization system can also provide a graphical environment in database systems in order to create more complex queries and data manipulation tools. Tioga-2 [39], , the extended and refined version of Tiago [40] (a data management to support scientific visualization applications), for instance, is a database visualization environment in which graphical representation of data can be manipulated using a set of basic operations. The majority of programming operations are performed by employing graphical representations. In Tioga-2, a set of primitive operations are successively composed to build practical database applications. Tioga-2 maintains incremental programming in which the programmer can observe intermediate results by getting an immediate visual feedback as a result of any modification.

Besides the issues in visualizing large data sets with respect to the size of the displays and the number of data items that can be represented on the screen, processing and finding a meaningful result in a large dataset as a time consuming and challenging process is another essential concern. To overcome these problems, work has been done to provide a graphical environment for an interactive data exploration by sending feedback to the user during the time of data exploration and analysis process. One research attempt that addresses the problem of the difficulty of finding a correct and meaningful answer to a query is FLEX [41]. FLEX is a cooperative and tolerant user interface that maintains the interaction between the user and the query specification analyzer. In the case that the query does not return a proper answer, the FLEX interface suggests or modifies the query to browse an acceptable result. VisDB [42], an interactive data exploration system, presents an approach to maintain filtering and approximate answers. It provides dynamic query modification and an immediate visual feedback; the former has a direct impact on the latter. When applying any changes in the query specifications, based on the amount of data in the new query result, the visual representation of data is changed which provides a feedback of the recent modifications. This takes into consideration, that in most data exploration and visualization systems, the visual representation has to be recalculated any time the query is modified and executed.

While the relational information visualization systems have been improved since early attempts done in 1980's, the definition of large amount of data shifted from tens of thousands to millions of data items. By data growing with accelerated speed, scaling visual presentations from tens of thousands to millions of data items has become an enormous challenge in information visualization and database management research area.

Initially an approach to conquer this issue was to visualize as many data items as possible by using each pixel of the display to represent a data item of the result set like IVEE system. However, this idea is not applicable in visualizing large data sets with millions of records. To overcome this problem, approaches are to reduce size of the data to be visualized, or to represent data at different levels of details. Shneiderman [18] believes that in designing an effective graphical user interface, his visual-information seeking mantra is the starting point: "overview first, zoom and filter, then details on demand." He presents the following primary but high level abstractions; overview (having an overview of entire data set); zoom (zooming into interesting data points to see more details); filter (filter out unwanted and irrelevant data); detail-on-demand (accessing the details of some interesting portion of data); relate (discovering and observing data items relationships); history (the possibility of reverse tasks and progressive refinement); and extract (extracting the query parameters or a subset of data). Some of these abstractions are used and developed in some early research attempts such as IVEE, IDES, and Visage [43], but still the idea of having an overview of large data sets remains challenging and overwhelming.

The filtering technique used in IVEE, for instance, presents a subset of data points in order to reduce the size of visualized data. IDES proposed by GoledSTEIN et al. [44] can be considered as some of the early work in defining the aspect of data granularity (multi-level data representation) and controlling the scope (region of interest). Based on their definition, in an interactive visualization environment, users can control the scope by selecting a specific portion of data, and the data granularity (the level of detail) by creating and decomposing aggregated values. Visage [43] also addresses this problem, and provides a more precise definition of interactive operations in data exploration. Visage provides filtering and partitioning in order to focus on interesting subsets of data (data reduction). By the means of drill-down and roll-up techniques (zooming in and out), the level of detail that has to be visualized can be controlled. Drill-down is defined as the process of decomposition of aggregated data into larger number of aggregated values of smaller data collections, while the roll-up technique refers to the process of re-aggregating smaller data groups to create new aggregation of larger data collections to reduce and summarize the visualized data. These aspects of roll-up and drill-down — also known as zooming in and out — have been used in much recent work to provide multiple levels of detail for representation in data visualization. VQE [45] (Visual Query Environment) adds more capabilities to direct manipulation and data exploration in Visage. VQE has the capability of storing exploration sessions (queries and visualizations) and reuse them on different datasets.

While many techniques were being developed to maintain the interactive data visualization, more general systems were designed to work with different data sources. DEVise [46] and Rivet [47] provide more general and flexible visualization and exploration environments for visual representation and manipulation of large data sets from disparate sources. DEVise supports exploration of large data sets from either local or remote databases by applying the query optimization techniques to deliver a robust visualization system. In addition to maintaining a flexible and simple visual environment, DEVise allows users to interactively share and explore visual data representations. Any modification made by one user in a data set is noticeable and available by other users observing the same data. While it provides exploring and handling large data sets from distributed data sources in a visual exploration, the user can drill down to browse a single data record (zooming in) for more detail. Following the same objective to develop a flexible and more general visual exploration system, Rivet provides an interactive data exploration environment that allows users to import arbitrary large data sets from different data sources. Rivet is flexible in that it supports user-defined transformations along with powerful standard operations such as filtering, aggregation, sorting, and grouping. Once data from other resources is imported and transformed in an internal database, each individual tuple is mapped to its visual representation using visual metaphors. A visual metaphor is responsible to encode data tuples to visual primitives on the display. Rivet also provides discriminating a subset of data using identifiers called selectors while data tuples are encoding to visual objects within metaphors. Multiple selectors can be defined in a metaphor to distinguish several subsets of data using different primitives associated to each selector. The idea of designing some systems such as Rivet is to develop effective and interactive visual exploration systems compatible with the iterative and unpredictable nature of data visualization and analysis of large datasets.

With significant work done to improve interactive data visualization systems, dealing with the large number of data items has still remained challenging and many research attempts have continued to be devoted to this problem. More detailed and comprehensive overview on interactive visualization of large datasets is presented by Godfrey et al. [11]. They provide a survey on three major areas in the visual exploration of large data sets: visualization techniques and related developed systems; query optimization techniques; and challenges in visual representation of data. One essential approach to deal with the issue of exploring and visualizing large datasets is data reduction. Data reduction can be performed either directly by the database system using available features and tools devised within the database system for data manipulation and transformation, or by visualization management tools. In database systems, the size of data can be reduced via sampling, aggregation, and filtering techniques [12]. When data sets are very large, analysis, manipulation, exploration, and visualization can be performed on a subset of data points instead of whole data sets. Sampling is one technique to statistically select a subset of data points in order to reduce the size of data in the visualization process. One concern in applying sampling technique on data is the possibility of loosing relevant and interesting data. Aggregation can be applied to the groups of data in order to reduce the number of data points to be visualized. The aggregated data points have same characteristics of the original data values and aggregate values can be defined as the summaries of the underlying data. Filtering can be described as selecting a subset of data with specific characteristics.

Aggregation as one of the most effective techniques in data reduction has gained significant attention in the relational data exploration and visualization research. Since aggregated data preserve the characteristics of original data, it provides some interactive operations such drill-down and roll-up. The user can also select the granularity of the aggregated data. When an overview (summarized data) can be expressive and effective enough, avoiding processing and analyzing individual data points improves the performance of the interactive visualization process. In database systems, aggregation is a query with a group function summarizing data via standard aggregation operations such as average, sum, and max. Fredrikson et al. [48] uses aggregation to simplify the display since fewer number of points (markers) is required to be visualized the aggregated data (data summaries). Their system provides two coordinates: aggregation; and detail. A user can have an overview of desired information in the aggregation coordinate and, at the same time, look at required details available in the detail coordinate.

Aggregation is a prominent technique in data reduction. However, aggregating large data sets is a time-consuming and costly process. The user has to wait a significant amount of time until the aggregation process is completed to see the final result, without any feedback in the meanwhile. In systems in which approximate answers are acceptable, an approximate answer can be provided for the initial query. The user can then refine the query after receiving feedback to explore and investigate a data set for desired patterns and knowledge. In aggregating large data sets, some approaches have been developed to address the problem of latency in answering aggregate queries. Hellerstein et al. [49] propose a new aggregation interactive interface called online aggregation. Online aggregation provides the possibility of observing the aggregate result progressively, and controlling the query execution whenever is needed on the fly. Some systems apply different data reduction techniques to scale down the processed data by considering system limitations and optimization techniques to improve the data visualization performance. Scalar [12] uses powerful features such as optimization tools in database management systems to improve the interactive visualization process by limiting the amount of data in the query result. This cap on data volume can be computed by statistically evaluating query plans to estimate the number of data required to be processed. Whenever the expected query result is too large to be efficiently and effectively visualized, data reduction techniques such as filtering, sampling, and aggregation are applied to summarize the result set.

Jugel et al. [50] propose M4 aggregation, a visualization-oriented data aggregation model, along with a visualization-driven query rewriting technique to dimensionally simplify time-series data reduction. When a query is issued over a high volume of time-series data, the query is rewritten in order to scale down the result set by applying data reduction techniques in the database. Systems like Scalar and M4 apply some parameters such as the display limitation and predictive techniques like query plans to reduce the number and the scope of data needed to be processed. VisReduce [51], in a new approach, uses MapReduce [52] as a powerful tool in large data processing. MapReduce is a powerful model to execute parallel and distributed algorithms over a set of workers (computers). Since VisReduce permits any query to be executed on an arbitrary group of workers, the processing time and the scale of the processed data sets can be accelerated by increasing the number of workers. As data is processed in a distributed and parallel manner, a feedback can incrementally be sent to the user while more data is processed and visualized. VisReduce provides a visualization environment with low latency by providing progressive feedback to the users; however, it does not define an overview of summarized data, not interaction operations such as zooming and panning to investigate and explore data.

One of the most essential contribution from the database community to data exploration and visualization has been to develop condense data structures that provide effective data reduction and multi-resolution representation techniques to supports interactive visualization of large data sets [53]. Elmqvist et al. [13] define a class of visualization techniques that represent a multi-scale structure using hierarchical aggregation which maintains interaction techniques. The authors represent a hierarchical aggregation in the form of a tree structure where non-leaf nodes are aggregated values and the leaves carry original data items. In this aggregated hierarchical structure, a non-leaf node (an aggregated value) can have one or more children and the root of the tree represents the whole data set. The aggregate tree structure can be constructed hierarchically in a bottom-up or top-down process. Bottom-up process starts with original data points, repeatedly applying aggregation on the subsets of original data or aggregate data until there is only one aggregate value. The top-down aggregation starts from the root (the aggregate value) and iteratively splitting the aggregate values until it produces leaves. Since aggregate values convey information about the underlying data, visualization techniques of hierarchical aggregation support the visualizing of aggregate data as well as original data points. One of the pioneering attempts in visualizing hierarchical data structures is from 1991. TreeMaps [54], a visualization technique, maps a complete hierarchy on to the display based on the space filling aspect in the form of rectangular area. This technique provides an overview of a large hierarchy that permits users to navigate any interesting area in the visual space. The authors assign a weight to each node to determine the size of the space the node occupies on the display called bounding box. However, this approach does not provide hierarchical data representation in a multi-resolution manner.

The OLAP cube [7] is another structure can be used in multi-resolution data visualization to reduce the size of the processed data and support various data granularity. Polaris [55] is the extension of Pivot Table [56]. It provides multi-dimensional exploration interface for database systems. Polaris supports a table-based graphical representation, but not the hierarchical structure of dimensions. It supports the data representation in each dimension independently. Later, Polaris was extended to effectively explore pre-summarized data cubes by supporting its hierarchical and multi-dimensional structure [57]. Hierarchical structures such as data cubes integrated with some data-reduction technique can maintain an interactive data visualization interface to explore large data sets in various granularities. imMens [15] relies on binned aggregation as its primary technique in data reduction, and the data cube structure to support interactive visualization of large scale data sets by precomputing multivariate data tiles. Multivariate data tiles are simply materialized database views. Binned aggregation, also applied by Hadley Wickham [16], is a technique to summarize data into a set of predefined bins. Binning summarizes the specified number of data points into bins (or intervals) represents an approximation of data. Binned aggregation represents data in multi-level resolution and maintains both global patterns and local features.

We propose inductive aggregation to build the hierarchical aggregate pyramid [58]. We employ aggregation as a data-reduction technique maintained as in some of systems described above. A practical and effective data exploration and visualization environment not only must be responsive in answering users' queries, but also must be smooth and rich to provide answers in desired level of detail in a multi-resolution representation model. The hierarchical characteristic of the pyramid structure along with aggregating data points incrementally supports interactive visualization of large data sets. The summarized data with the same scale of canvas is sent to the user. Additional operations to explore data in more detail (resolution) can be maintained by interconnection between different resolution levels of the aggregate pyramid hierarchy. Following Schniderman's mantra, the aggregate pyramid provides overview, zoom in, and detail on demand. The precomputaion of the aggregate pyramid can also address the latency of aggregate queries response, since summarized and precomputed data provides an efficient and effective interaction in database exploration and visualization.

Chapter 3

Inductive Aggregate Pyramid

One of the most crucial and important aspect of data analysis and visualization is a fast and smooth (real-time) interaction. Visualizing large datasets makes this problem more challenging when the dataset is too large to fit the screen or even in main memory. Various techniques have been developed to deal with the issue of large dataset exploration and visualization. Some systems employed techniques to provide approximate answers to the queries using some predictive parameters in relational database systems [12, 59] and data reduction techniques such as filtering, sampling, and aggregation to eliminate unwanted and irrelevant data points to summarize and scale data based on the screen limitations [12].

One approach to reduce the size of data in order to develop an interactive visualization environment is to have the data reduction techniques maintained by a database system to provide filtering, sampling, and aggregation. One important concern in data reduction is that some relevant and interesting data might be discarded during the data reduction process. Sampling and filtering techniques, for instance, select a subset of data in order to scale down the dataset. However, by selecting some interesting and desired portion of data in data reduction by applying filtering or sampling, there is a possibility to lose relevant and interesting data. Aggregation, on the other hand, is a data reduction technique that processes all required data by applying aggregate functions on predefined groups of data values. Unlike sampling and filtering, in the aggregation operation, all desired data points are processed to form aggregate values and the characteristics of the original data values can be preserved while summarizing the underlying data. Furthermore, by re-aggregating group aggregates to have larger aggregate data collections, different granularity of data can be provided by this technique.

Our objective is to propose and develop a data structure containing multiple levels of detail to maintain fast and real-time data exploration and visualization of large datasets. The proposed structure scales to collections of data of any desired size. In order to have data at multiple levels of detail, an aggregation technique is applied to summarize the dataset. The original and summarized data can be organized in a hierarchical, compact, and successive structure which is often called a data pyramid.

3.1 Data Summarization

The purpose of data summarization is to reduce the size of data while keeping the properties and characteristics of the original data points. Summarized data delivers the same concept (knowledge) and information as the original data values. In other words, in data exploration of summarized data, reaching the same explanation of original data values is expected. The data summary is a reduced version of a dataset in a comprehensive and semantically meaningful form.

To scale down a data set, data reduction techniques solely may not be sufficient since we want to preserve some essential key elements of the original data in the compact form. Some other factors have to be considered along with data reduction techniques to translate the original dataset to the reduced version. We introduce an approach to summarize data called *inductive aggregation* to support data summarization using data reduction techniques along with other methods to preserve the original data characteristics. To summarize sufficiently geometric data, for instance, we use techniques to preserve the original data properties such as the locality of data points.

To have interactive and efficient data visualization, we need to be able to explore data in an efficient and real-time way. However, when it comes to a huge dataset with billions of tuples, exploring data is a time-consuming and costly process. Even in the cases when an overview (summarized data) is provided, the user may ask for further detail by "zooming in" to a subset of the data or want to change the viewpoint (panning) — moving to an adjacent portion of data — to explore other areas. Therefore, more data has to be retrieved and processed to provide a data summary at the desired granularity to provide the overview and to browse more detail. One way to make this process faster and more efficient is by preprocessing the data and making it ready to explore by generating multiple projections and representations of the original dataset in advance, providing the summaries of data at different granuralities. Having the multiple representations of pre-computed and summarized data gives us the possibility of exploring data in different resolutions of detail without the necessity of processing over the whole dataset for every request.

3.2 Hierarchical Data Structures

In spatial database systems, hierarchical data structures such as quadtrees [26] have been proposed to sort the data based on their occupancy in the space known as spatial indexing. These hierarchical data structures provide efficiency by facilitating some operations on data. Quadtrees and pyramids are two popular hierarchical data structures in image processing and spatial database systems.

Pyramids can be defined as hierarchically organized and interconnected structures. By providing interconnections between different levels from detailed data points to more global information (containing summarized and aggregate data), the pyramid hierarchy maintains computational efficiency by supporting optimized and efficient algorithms for both constructing and querying summarized data. The idea of using the pyramid to perform search on an image, was introduced initially by Tanimoto, et al. [23]. The search operation as a top-down process is initiated at a low resolution level (the top of the pyramid) with progressive refinement by processing more detail at higher resolutions (lower levels of the pyramid). Kriegel et al. [24] propose a pyramid technique as a novel way to index high dimensional data sets. Their primary idea is to map a multi-dimensional data space to one-dimensional data points in order to be supported by an efficient index structure such as a *B+-tree* to improve the performance especially for range-query processing. The PT (Pyramid Technique) [24] partitions high dimensional data spaces into two-dimensional pyramids with the top at the center point of the data space. Each two-dimensional pyramid is then transformed to one-dimensional data points. The hierarchical multi-resolution structure of the pyramid makes this model a popular structure for indexing high dimensional data spaces in spatial data systems and image processing.

Quadtrees and pyramids are two prominent structures in multi-dimensional indexing in spatial database systems. However, there are some distinctions between these two structures. The pyramid is a multi-resolution representation while the quadtree is a multi-variant structure [26]. The hierarchical multi-resolution structure of the pyramid maintains a sufficient configuration to summarize data at multiple levels of detail. The multiple representations of summarized data can then provide the approximate answers to queries. While the pyramid representation provides a summarized data representation to give approximate answers to the queries, the quadtree model can produce exact answers. Moreover, pyramids and quadtrees differ in supporting the various types of spatial queries in spatial database systems [60]. Samet et al. [61] define two types of spatial queries: location-based and feature-based queries. While location-based queries can simply be answered in a quadtree structure, there is more complication in answering feature-based queries, since quadtrees are indexed based on spatial occupancy and do not provide indexing by features. Comparing the characteristics of quadtrees and pyramids, pyramids are better structures to answer feature-based queries. To determine the existence of a feature in the pyramid structure, lower-resolution data (higher levels of the pyramid) can be examined. This reduces the response time of the query by negating the need to examine large portions of data to check for the presence of a feature. Moreover, building the pyramid structure is a bottom-up process in contrast to the construction of quadtrees which is a top-down operation. One of the more important distinctions between quadtrees and pyramids can be seen in their non-leaf nodes. In quadtrees, unlike pyramids, a non-leaf node does not contain data or information about the underlying data points and nodes [62].

The pyramid as a multi-resolution representation of data has advantages compared to other hierarchical structures [63] for our application. They provide interconnection between local (more detailed data) and global features (region information or aggregate data) by mapping global features into local ones and vice versa. Because of the hierarchical nature of the pyramid, a recursive decomposition, similar to the divide-and-conquer principle, can be employed to reduce the cost of operations and improve performance [63, 64].

3.3 Multi-dimensional Ordering

Many analytical problems rely on processing cellular data (bins) in the form of a multi-dimensional gridded space. For efficiency, this requires a single-dimension cellular ordering to reduce the multi-dimensional space to a one-dimensional domain, as pointed out in [62]. This spatial ordering simplifies the complexity of analysis procedures by exploiting previously designed data structures and techniques. Mark and Goodchild [65] define an ordering as a sequence of unique consecutive key values assigned to a set of n distinct spatial objects (e.g., bins or points) from 0 to $n-1$. An ordering maps a multi-dimensional plane to a one-dimensional space in order to simplify analytic processing while preserving the spatial locality of data objects. By storing spatial objects in a linear order, the sequential processing on a consecutive list of data objects can be performed in an efficient way. For two-dimensional data, each bin can be partitioned into 2×2 cells/bins called quadrants. Mark [30] defines the three following properties for cellular data objects partitioned onto a $2^n \times 2^n$ grid where n is large enough that no two points (with distinct x and y valuess) fall into one bin;

- An ordering is **continuous** if and only if each two data objects with successive order keys are spatially neighbors.
- An ordering is **quadrant-recursive** if the keys assigned to the subquadrants of a cell (quadrant) are consecutive in an order. In a $2^n \times 2^n$ grid with ordering levels from 0 to n, any level L ($0 \leq L \leq n$), has $2^L \times 2^L$ cells with consecutive keys from 0 to 2^{L-1}.
- And finally, an ordering is **monotonic** if and only if for any fixed coordinate x, the keys monotonically vary with respect to y, and vice versa. An ordering is positive monotonic if the value of keys increases by increasing the value of one dimension

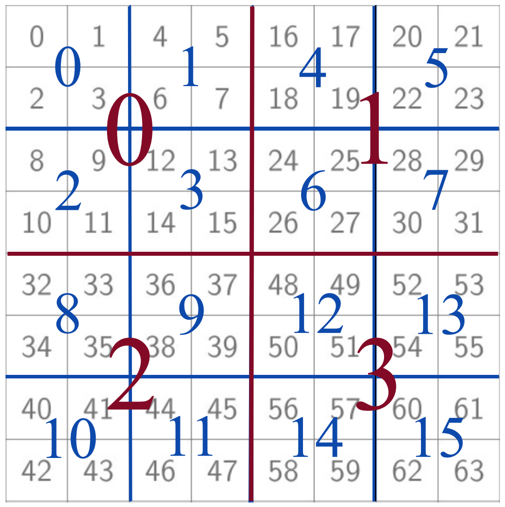


Figure 3.1: Quadrant-recursive property of Z (Morton) order

(e.g., x) while the value of other dimension (e.g., y) is constant.

The quadrant-recursive ordering property has garnered significant attention because of its compatibility with popular data structures such as quadtress. A multiple-level ordering can be recursively constructed based on this property [30]. In a quadrant-recursive ordering, a rectangular region (quadrant) is recursively decomposed to sub-quadrants in a consecutive order. In other words, in a successive step, a quadrant is divided into four sub-quadrant;, each sub-quadrant is then subsequently partitioned into four sub-sub-quadrants; and so on. Multiple levels of a quadrant-recursive ordering can be constructed by applying a recursive algorithm.

Multi-dimensional orderings also known as space-filling curves (Peano curves) [66] were proposed for the first time as early as in 1890, by mathematician Giuseppe Peano [29]. A space-filling curve (in our discussion of multi-dimensional orderings) continuously maps a multi-dimensional space into a one-dimensional domain by assigning a unique number to every point in the multi-dimensional space. As a result, distinct points are mapped to different numbers. In the space filling curve technique, two points that are spatially adjacent are likely to be close in the linear ordering resulted by space filling curve mapping. The idea of mapping the multi-dimensional data points to the one-dimensional data values can be used in spatial indexing. The one-dimensional values resulting from a space-filling curve can be efficiently indexed by an index structure such as a *B+-tree* to improve performance especially for the spatial-range query processing. In many spatial data analyses, the query processing is more efficient if spatially adjacent data points are contiguously stored on the disk.

Morton order [20] and Hilbert order [67] are two popular multi-dimensional orderings that have received considerable attention in image processing and spatial database systems. They hold the three properties as described by Mark [30]: continuity; quadrant-recursive ordering; and monotonicity. Considering their quadrant-recursive characteristics, these two ordering are compatible to quadtree-form structures. The idea of assigning a linear ordering to spatial data objects in a multi-dimensional space while preserving the spatial contiguity of the keys has been applied to quadtrees, known as linear quadtrees [26,27,68,69]. The term linear quadtree, introduced by Gargantini [27] in 1982, indicates a class of data structures where leaf nodes are representing by a set of sequential numeric keys. Figure 3.1 presents the quadrant-recursive property of Morton order for three levels.

We adopt Morton order in this research to construct and index the aggregate pyramid; this ordering is also known as "Z order" [70, 71] and as "N order" [72, 73]. Morton order [20], introduced by G.M. Morton in 1966, is a space-filling curve technique that maps multi-dimensional data space to one-dimensional domain while preserving the spatial properties of data points. In this ordering, the key value (the morton number) of a data point is calculated by interleaving the binary representation of its coordinates' values. In calculation of the Z-order values, the process of interleaving the data point coordinates' bits is called the shuffle of coordinates. In the rest of this study, we use the term "Z-order" to address Morton multi-dimensional ordering.

3.4 Aggregate Data Pyramid Model

Employing database query optimization and data reduction techniques in data visualization requires a tightly coupled system with integration of database and visualization systems. Besides the powerful features for data reduction and optimization tools, scalable and compact data structures can be developed in database systems to support interactive data exploration. OLAP cubes and pre-computed materialized views, for instance, provide intermediate pre-aggregate results that facilitate query evaluation and processing. To browse interactively a large volume of data, imMens (Real-time Visual Querying of Big Data) [15] employs multidimensional data cubes along with binned aggregate techniques to develop a multi-resolution data structure supporting interaction. Zuotao Li et al. [74] propose a pyramid model to browse interactively and discover content-based knowledge to support statistical range query processing. Their data pyramid stores relevant and desired information in condense and approximate form. A query answer can be prepared by querying the data pyramid rather than the whole dataset.

We propose a compact multi-resolution structure that facilitates the process of data exploration and visualization by providing the summarized data at different granularities. In order to have a multi-resolution representation of data summaries, we propose a multi-resolution compact structure called the inductive aggregate pyramid. This structure provides a hierarchy to store and process the data at multiple levels of detail. To have a multi-resolution data pyramid, data has to be preprocessed and pre-computed in advance. First, the raw data has to be processed to form the base of the pyramid. Then, the base data is reprocessed to form the next stratum, which is the summary of the underlying base stratum of the pyramid. This process can be repeated until computing the apex stratum of the pyramid (the top of the pyramid) — which is a single bin represented an aggregate of the whole dataset — inductively and progressively.

The motivation for our aggregate pyramid model is to exploit a database system to support visualization tools and techniques to explore and represent a large volume of data. The hierarchical structure of the data pyramid helps users efficiently to narrow down to a subset of the data they are interested in. The pre-computed and preprocessed multi-resolution data represented in the hierarchical form of the pyramid can then be considered as a data source for visualization and data mining tools.

3.4.1 Progressive and Interactive Visualization

One major obstacle in visualizing very large datasets is the massive amount of data required to be processed and represented. This same problem has been addressed in image processing and visualization to render and transfer very large images. Large images need more storage, and the transmission of digital images with high quality can be quite slow. Moreover, the incompatibility of the image resolution with the display size imposes additional difficulties in image transmission and rendering. Encoding images in a compressed form, transmitting images progressively, and/or having multiple copies of an image with various qualities in order to scale to a resolution compatible with the display size are imperative solutions to address these issues in image processing. The JPEG standard [75] maintains both lossy and lossless image compression. In lossless compression, the original image can be decoded and recovered in entirety from its compact form without losing any information. Lossy compression, on the other hand, produces images in more compact form with smaller size in comparison to the lossless technique but with imperceptible and acceptable loss of information. The JPEG standard also proposes both progressive and hierarchical modes for image compression [75, 76].

Progressive image transmission is known as progressive mode in the JPEG standard. In the progressive mode, an image can be transmitted in multiple passes. In the first pass, an approximate version of the image with low quality and size is sent to provide a quick overview in order to have a fast and real time interaction. In subsequent passes, the image quality is progressively increased by sending more bit information. Ultimately, an

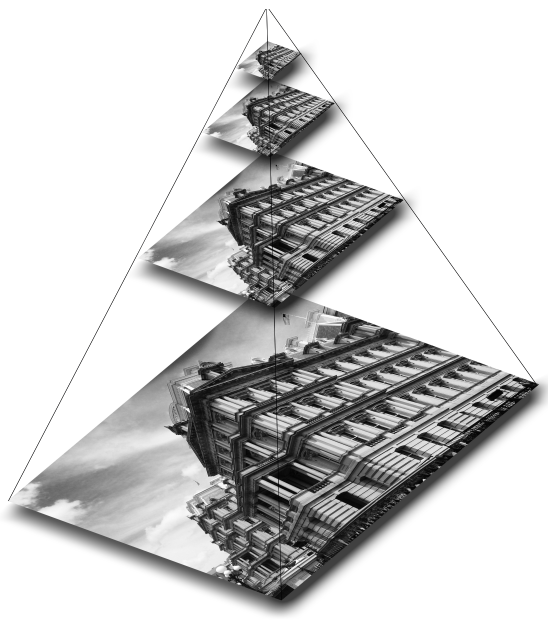


Figure 3.2: Four level image pyramid

accurate approximation of the original image is constructed by the progressive refinement of the initial low-quality version of the image. To have progressive image transmission, some techniques called progressive compression techniques are applied to encode a low-quality version of the imge . The encoding can be performed on either the original image or any intermediate lower quality version of the image to create progressively various versions of the image with different resolutions.

The JPEG standard [75] also supports another form of progressive transmission known as hierarchical mode; this is multi-resolution representation of an image. In this mode, the image is decomposed into a pyramidal structure of multi-resolution copies of the image where the top of the pyramid represents the lowest resolution of the image and the base provides the image version with highest resolution (Figure 3.2). In the hierarchical mode, the image is encoded to the lower resolutions using low-pass filters. As a result, images with higher resolutions successively represent more detail.

The resolution, in the hierarchical mode, is progressively increased at each step producing a sequence of different versions of the image from high to low resolutions. Each lower level version with higher resolution of the pyramid is used as a source to create a very next upper level with lower resolution. This method can be applied whenever the resolution of the requested image is very high such that it cannot be displayed on a low resolution screen. Like progressive transmission mode, the hierarchical mode also provides the progressive presentation and the encoded image with different resolutions can be progressively transmitted. This technique can effectively be used in applications supporting a variety of representations and different screen sizes.

3.4.1.1 Compression Techniques

Progressive mode is employed to improve the user experience of viewing and interacting with images on the web. JPEG2000 [22] is an image compression and multi-resolution format that provides multiple representation of an image by its compression process. This feature also provides progressive transmission in JPEG2000 [22]. For compression, images are decomposed into progressively lower resolution approximations of the full size image. The approximations are ordered from highest to lowest resolution.

The JPEG2000 [22] image compression standard is based on the Discrete Wavelet

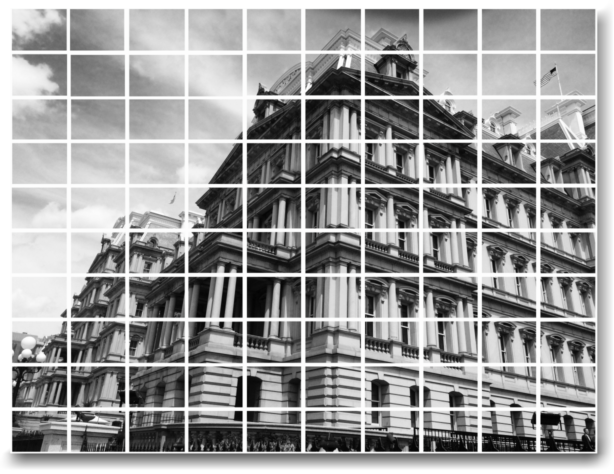


Figure 3.3: the tiled image

Transformation (DWT) [77]. An image in JPEG2000 can be defined by image tiles which are rectangular non-overlapping blocks. In the tiling process, a source image is partitioned into equal rectangular regions called tiles (Figure3.3). These tiles are considered as independent images for the compression process and some operations such as the wavelet transform can be performed on each tile. Since each tile is compressed individually, a part of the image can be independently reconstructed by decoding the correlated tiles of that region. In the tiling process, the dimensions of tiles are powers of two (with the exception that the tiles' dimensions on the image boundaries that may not be the powers of two).

Resolution	Average	Detail Coefficients
2	<7 5 2 6 >	
1	<6 4 >	<1 -2 >
0	<5 >	<1 >

Table 3.1: One-dimensional Haar Wavelet Transform

Wavelet Transformation

The wavelet transform is a computational technique used in signal and image processing applications and can be applied to represent an image at different levels of detail. Image tiles are encoded and compressed into different decomposition levels employing the wavelet transform technique. This method is used in the JPEG2000 standard as wavelet-based compression algorithms [78]. Image compression techniques use two-dimensional Haar transform that is a generalization of its one-dimensional technique. Haar functions were introduced by the Hungarian mathematician Alfred Haar in 1910 [79].

One-dimensional Haar Wavelet Transform One-dimensional Haar Wavelet Transform [80] can be applied on a one-dimensional image. The following vector can be considered as a one-dimensional image with four-pixel resolution; $< 7\ 5\ 2\ 6 >$. Table 3.1 represents the wavelet transform computation in a Haar basis.

The four-pixel resolution image (Table 3.1) contains the original image information that represents the high resolution of the image. The second row, the lower resolution,

9	15	11	17
4	6	9	7
3	11	6	8
1	3	2	4

(a) 16-pixel 2D image

13	-1	-3	-3
6.5	-1.5	-1	1
7	0	-4	2
2.5	-0.5	-1	-1

(b) Wavelet transform of each row

7.25	-0.75	-2.25	-0.75
2.5	0.5	-0.25	-0.25
3.25	0.25	-5	-2
2.25	0.25	-1.5	1.5

(c) Wavelet transform of each column

Figure 3.4: Standard decomposition

is the result of pairwise averaging of the four-pixel resolution. Similarly, the last row, the lowest resolution, is computed from pairwise averaging of the two-pixel image. Since some information is lost in averaging, we need detail coefficients to recover the higher resolution pixels from the lower ones. The first coefficient in the two-pixel resolution is 1 which conveys that the first average value of the two-pixel image is one unit less than 7 and one unit more than 5 in four-pixel resolution version. The coefficient 1 and the average pixel 6 can be used to recover the first two pixels of the original image. The vector $< 5\ 1\ 1\ \text{-}2 >$ represents the final transformed image. One-dimensional Haar Transform can be generalized to two-dimensional Haar Transform.

Two-dimensional Haar Wavelet Transform There are two types of two-dimensional wavelet transforms [80]; standard decomposition and nonstandard decomposition. In the standard decomposition of an image (Figure 3.4), one-dimensional wavelet transform is first applied to each row image data that results average and detail coeffi-

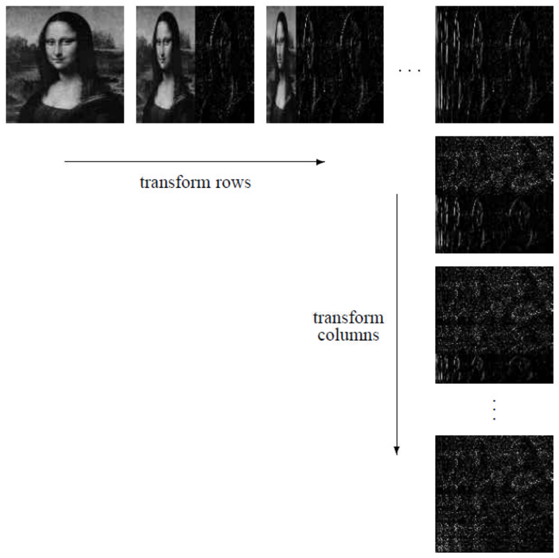


Figure 3.5: Standard decomposition of an image [80]

cient values in each row. Then, the transform function is applied to each column. At the end, in the standard decomposition, we have coefficients and one average value which is the average of the whole image. Figure 3.5 is an example from “Wavelets for Computer Graphics: A Primer” [80] that shows the standard decomposition of a two-dimensional image.

Figure 3.6 displays the nonstandard decomposition of a two-dimension image matrix with 16 pixels resolution. The nonstandard decomposition applies pairwise averaging and differencing on the image data in each row horizontally and then it computes pairwise averaging and differencing of the result (average values) in each column vertically. Pairwise averaging and differencing on rows and columns of the cells containing average data values continues until there is just one square on top left of the image matrix that

9	15	11	17
4	6	9	7
3	11	6	8
1	3	2	4

(a) Two-dimensional image with 16-pixel resolution

12	14	-3	-3
5	8	-1	1
7	7	-4	2
2	3	-1	-1

(b) Horizontal pairwise averaging

8.5	11	-3	-3
4.5	5	-1	1
3.5	3	-4	2
0.5	2	-1	-1

(c) Vertical pairwise averaging on averaged data values

9.75	-1.25	-3	-3
4.5	5	-1	1
3.5	3	-4	2
0.5	2	-1	-1

(d) Horizontal pairwise averaging on averaged data values

7.25	-1.25	-3	-3
2.5	5	-1	1
3.5	3	-4	2
0.5	2	-1	-1

(e) Final wavelet transformed image

Figure 3.6: Nonstandard decomposition

represents the average of the whole image data. Figure 3.7 is an example of nonstandard decomposition of a two-dimensional image from “Wavelets for Computer Graphics: A Primer” [80].

3.4.1.2 Data Compression

We can take advantage of the previous techniques to define a multi-resolution dataset constructed by an inductive aggregation process to compress data in a multiple represen-

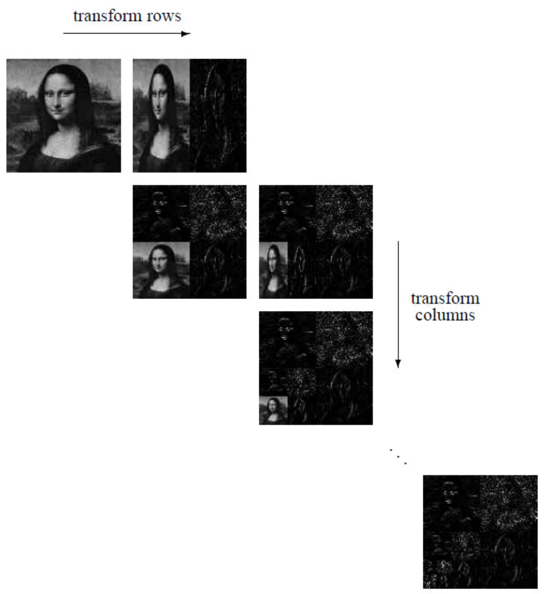


Figure 3.7: Nonstandard decomposition of an image [80]

tation format. The first aggregate result retrieved from the database can be considered as the source data similar to the original image in the compression process. In the source data, each tuple is considered as a bin. Figure 3.8 depicts a multiple resolution two dimensional dataset. The 16×16 two-dimensional matrix represents a dataset of resolution 4 with $2^4 \times 2^4$ cells. Each cell in the matrices represent a bin. Tiles in a two-dimensional dataset can be referred to a bin or a group of bins.

Let the 16×16 dataset form the highest resolution of our data pyramid. The source tuples can be composed to 64 bins which each bin includes the aggregate value of the corresponding quadrant (four bins) of the source tuples. The average data values in each quadrant is then processed to create the bins of the next lower resolution in the hierarchy

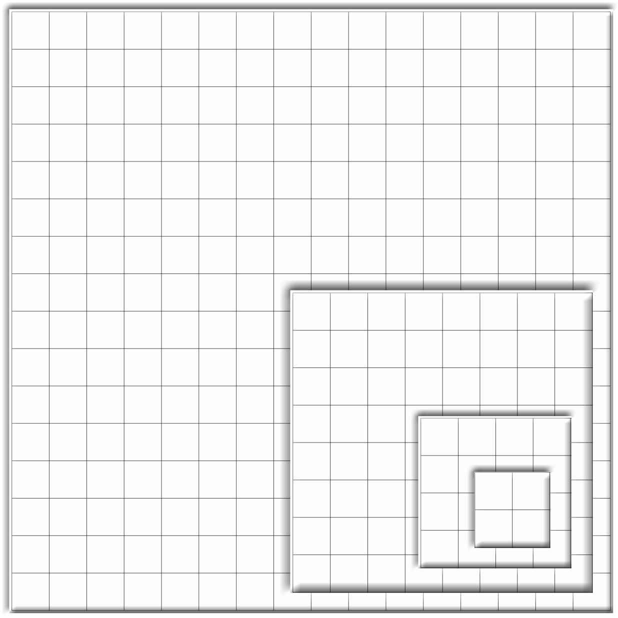


Figure 3.8: A multi-resolution dataset

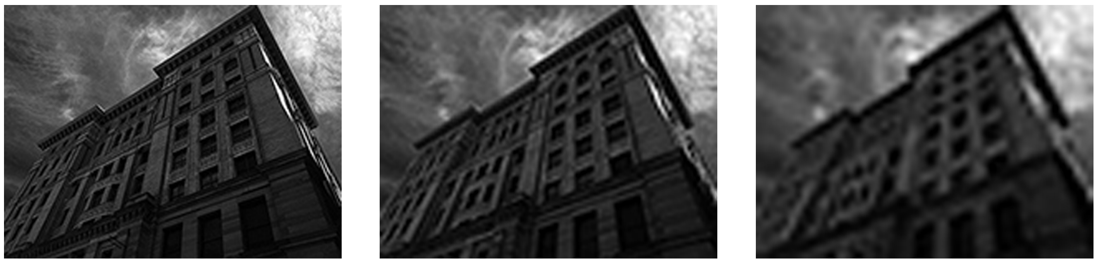
3.8. This process continues until producing the lowest resolution which is made of the one bin containing an aggregate value that represents the average of the whole source tuples.

In fact, by having a hierarchy of data resolutions as a source for data visualization, whenever the requested data is very large so that it cannot fit the canvas; a data overview (a data summary) can be transmitted and visualized in the visual space. In order to reduce the size of the visualized data, some data reduction techniques are required to apply on the original requested dataset. While the lower resolution data is sent to the user, the user may ask for more detail which needs to retrieve more data from the target relation. In this case, the user has to wait for the system to compute the new request with the desired resolution which increases the response time. One way to overcome this problem is to precompute and provide data at different resolutions and retrieve the requested data from precomputed and preprocessed hierarchy of different data resolutions in an interactive manner. A hierarchical data structure such as the pyramid model [74] can be used to precompute and store the data at various resolutions of detail.

We assume a dataset is too large when the number of the data points is bigger than the number of pixels of the display. By precomputing and storing data in a compact multi-resolution form, to visualize the requested data, we can send a slice of lower resolution data tuples to the visualizer compatible to the display size. In order to build a hierarchical data structure with different levels of resolution, In the first step, the tuples with original data values retrieved from the target relation has to be processed to form the highest resolution version of the dataset. Next, a lower resolution slice is built of aggregate tuples which represents the average resolution of the previous slice; and so on. If the user asks for more detail by zooming or sizing up, the system sends the tuples of the relevant higher slice of resolution. If the user wants to observe a different part of the image, the system applies the appropriate range selection to retrieve the related tuples from the corresponding resolution slice.

3.4.2 Inductive aggregate pyramid structure and specification

The image pyramid, a multi-resolution data structure, represents a $2^d \times 2^d$ image as a sequence of copies (two-dimensional arrays) of an original image at different levels of



(a) 204×153 image (b) 102×77 image (c) 51×39 image

Figure 3.9: Hierarchical resolution of an image

resolution. The pyramid P consists of $d + 1$ levels (from d to 0) called the depth of the pyramid. The P_i represents the level i in the pyramid P which has a half of the resolution of P_{i+1}. Each level is composed of equal rectangular cells (bins). The P_0 is the top layer in the pyramid which consists of one bin representing the average of the whole image. Figure 3.9 represents an image in three different resolutions in which the image with lower resolution has the half of the quality of its previous image with higher resolution.

The data pyramid, similarly, is a representation of a $2^d \times 2^d$ data in the form of a stack of two-dimensional arrays containing the data at different resolutions. Given a two-dimensional dataset, the data tuples is considered as bins representing the rectangular cells in the base stratum of the data pyramid. As mentioned, the construction of the data pyramid is a bottom-up process. The data pyramid is generated from the base stratum (preprocessed data values from original dataset) by recursive composition of data tuples or bins. The recursive composition can be applied by performing the aggregate functions on each quadrant (four bins) of the base stratum to build the immediate next stratum with the lower resolution. Each P_i is generated from the immediate lower stratum P_{i+1}.

This process continues until the last stratum P_0 is constructed which contains a bin with one aggregate value. Sub-queries can also be performed on the data pyramid to provide different data resolutions to be visualized in response to the user's request in an interactive visualization.

3.4.2.1 Inductive Aggregate Pyramid

The inductive aggregate pyramid is an appropriate structure to be used for an efficient data exploration process. To explore and visualize a dataset, the database system materializes the pyramid from the given dataset, and indexes the pyramid by stratum (the resolution level) over the Z-order [20] assigned to the tuples in the precomputing process to build the base stratum of the aggregate pyramid. To explore the data at different resolutions, the queries of interactive operations over the aggregate pyramid is managed by the database engine. Visual interactive operations such as panning and zooming are efficiently supported by the data pyramid hierarchy. The aggregate pyramid supports range queries over a rectangular region using filling space curve techniques such as Z-order, which provides a real-time and efficient interaction. In other words, in a data visualization process, the aggregate data pyramid provides database support to manage and process very large datasets in an interactive manner.

The aggregate pyramid is made of various levels of data resolution (strata) containing aggregate values called strata. The base of the pyramid represents the stratum with the highest resolution of data. The raw data has to be processed and aggregated to form the base stratum. The higher strata represent successively lowest resolutions of data. In a two-dimensional aggregate data pyramid, the base of the pyramid is the rectangular two-dimensional data space. The dimensions are not necessarily equal but they have to be powers of two. In this study, we consider the dimensions equal, and (without loss of generality) the power of two.

Each stratum of the aggregate pyramid consists of bins. A bin is a tuple of aggregate values and related data such as dimensions, the stratum number and the Z-order (Morton order) [20] value. We define the aggregate window as 2^d bins (where d is the number of dimensions) to be aggregated to form the super-bins in the next higher stratum with lower resolution. All 2^d bins in the aggregate window of the current stratum are associated with a super-bin (a single bin) from the next higher stratum. In inductive aggregation for a two-dimensional dataset, aggregate windows are non-overlapping square windows of 2×2 (2^2) bins also called aggregate quadrants or data quadrants. Therefore, the aggregate pyramid is a right pyramid with its apex directly above the centroid of its base. Each four bins (a data quadrant) in the base stratum and other intermediate layers are associated with at most a super bin from the very next higher level and the single bin in the apex is correlated only to the four bins (one quadrant) from the stratum below the apex.

Z-order [20] is assigned to tuples (bins) at each level of data resolution. As a result, tuples at each stratum can be defined by their Z-order and their stratum number (level of resolution). By having the tuples in this order, we can apply aggregate functions on each group of four adjacent tuples (consecutive data points in the Z-order) representing aggregate windows (aggregate quadrants) in two-dimensional (and, likewise, for higher dimensional) data. The lowest resolution has one bin (tuple) that represents the aggregated value of all data points.

3.4.2.2 Inductive Aggregation

We define the inductive aggregation to pre-compute data in order to build the aggregate pyramid to represent the multiple projections of data summaries. In the following discussion, we describe the concept of inductive aggregation to build the aggregate pyramid for two-dimensional datasets which generalizes naturally to higher dimensional or one-dimensional datasets. Inductive aggregation processes and summarizes data progressively, starting from the base of the pyramid. This technique is applied on a group of four bins (aggregate quadrant or aggregate window) that represent the four locally adjacent points in a plane by Z-order. The inductive aggregate technique consists of two phases.

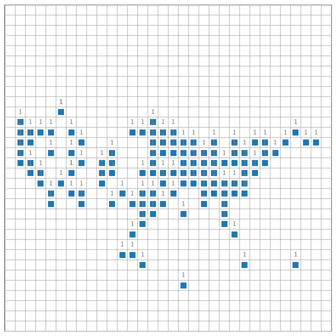
- The **base-aggregation** phase that processes and aggregates the initial raw data to form the base stratum of the data pyramid.
- The **re-aggregation** phase that iteratitively and progressively applies the aggregate functions over the aggregate windows (data quadrants) containing aggregate values (bins) to build the super bins of corresponding strata initially started from the base stratum in order to form the aggregate pyramid.

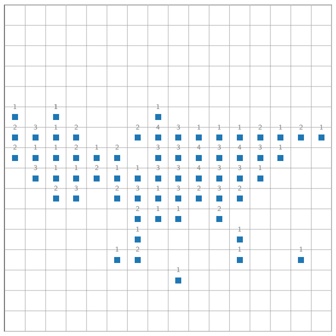
We devise how to aggregate hierarchically and efficiently very large datasets. We define inductive aggregation as a set of techniques and principles to build the aggregate pyramid in an efficient and progressive manner. In the base-aggregation phase, we determine the procedure of processing and aggregating the initial raw data to form the base of the pyramid. In this phase, a distinct number (Z-order) is assigned to each tuple in the base. Let B_n be the number of bins in the base stratum P_n retrieved from raw data, which forms a $2^n \times 2^n$ grid. In the re-aggregation phase, we iteratively apply the aggregate functions over the aggregate windows (data quadrants), which contain the aggregate values (bins) to build the super bins of corresponding strata initially started from the base stratum, in order to form the aggregate pyramid. To build the next stratum P_{n-1} from the base, an aggregate function is applied on aggregate windows of stratum P_n. By assuming that each of the $2^n \times 2^n$ bins of P_n have data from the raw dataset assigned to them, the size of stratum P_{n-1} is then $\frac{1}{4}B_n$. Successively, the intermediate stratum P_i $(0 \leq i \leq n)$ is constructed by employing the aggregate function over aggregate windows of the previous stratum P_{i+1}. Consequently, the number of bins B_i of stratum P_i is $\frac{1}{4}B_{i+1}$ (or $\frac{1}{4^{(n-i)}}B_n$). The total number of bins B_p in the two-dimensional aggregate pyramid is

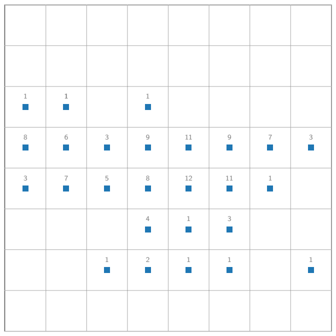
$$B_p = B_n + \frac{1}{2^2}B_n + \frac{1}{2^4}B_n + \frac{1}{2^6}B_n + ...$$

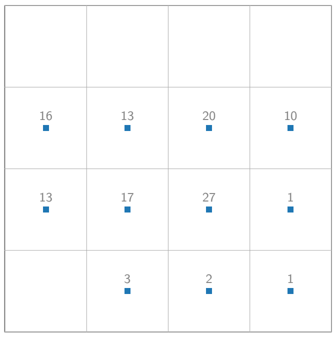
$$B_p = B_n \sum_{i=0}^{n} \frac{1}{2^{2i}}$$

$$B_p = 1\frac{1}{3}B_n$$

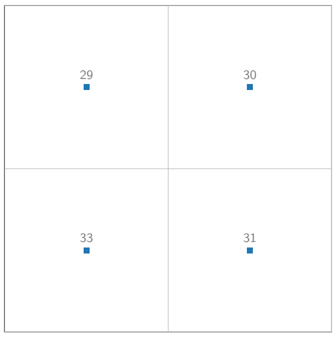

(a) Aggregate data 32 × 32


(b) Re-aggregate data 16 × 16

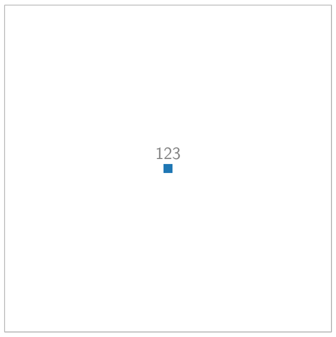

(c) Re-aggregate data 8 × 8



(d) Re-aggregate data 4 × 4



(e) Re-aggregate data 2 × 2



(f) Re-aggregate data 1 × 1

Figure 3.10: Six-level inductive aggregation over a sample dataset

Standard aggregate functions including *mean*, *count*, *sum*, *min*, and *max* are supported by this inductive-aggregation technique. In summary,to construct inductively the aggregate pyramid, first, we aggregate the raw data from the dataset to process and construct the pyramid's base stratum. Next, we iteratively re-aggregate quadrants (aggregate windows) containing aggregate values from the previous stratum to form the very next higher stratum with lower resolution.

Figure 3.10 presents an aggregate pyramid with six strata (from 0 to 5). In the base-aggregation phase, the base stratum (stratum 5) is constructed from the original dataset which possesses $2^5 \times 2^5$ bins. The non-empty bins (tuples) represent data points in the two-dimensional space with data value 1. In re-aggregation phase, the aggregate function *count* is applied over aggregate windows (2×2 bins) to build the next stratum, stratum 4 with $2^4 \times 2^4$ bins. The aggregate function is again employed over the aggregate values in each aggregate window of stratum 4 to build the very next stratum (stratum 3). The re-aggregation process continues until the top of the pyramid, stratum 0, is constructed that represents the total number of data points in the original dataset.

3.4.3 Building the Aggregate Pyramid

In construction of the aggregate pyramid, the first step is building the base stratum (the base of the pyramid). The cost of the base stratum construction relies directly on the size of the underlying dataset containing the raw data, and other factors such as existence of the indexes. The presence of indexes can facilitate the procedure of processing and aggregating the raw data into the bins. The bins of the base stratum can be computed in an order such as Z-order (Morton order) [20]. The Z-order space-filing curve technique is a one-dimensional, linear ordering for any multi-dimensional data, with respect to the dataset's dimensions. The computed tuples (bins) including aggregate values of the base stratum are then written sequentially to the disk, sorted based on the Z-order. Each stratum is built and materialized from the previous stratum, starting from the base up to the top of the pyramid. The last stratum (stratum i, i from n to 0 where n is the depth of the pyramid) is sequentially scanned once to construct the next stratum $i-1$. Since bins in stratum i are ordered by Z-order values, the aggregate functions can be applied on a group of four adjacent bins (the aggregate window) in the sequence to produce the corresponding super-bin in stratum $i-1$. The bins in stratum $i-1$ are also then sequentially stored on the disk sorted by Z-order values.

To build the pyramid, we need to determine how deep the pyramid needs to be computed. The base of the pyramid made of aggregate values is built by applying any aggregate function (sum, average, etc.) during the base-aggregation phase. Next, we need to define a depth which is meaningful and coherent to construct the base stratum from that depth. If the pyramid is too deep, in the lower strata, many bins could be empty. On the other hand, if the pyramid is not deep enough, it may not be able to provide required detail in the process of interactive operations such as zooming in or sizing up. We can build the base stratum as deep as possible to provide enough detail to build the next stratum. We want the next stratum after the base to have a proper and meaningful bin size (number of bins), which we name the natural depth. This should be deep enough that, if the user wants to narrow down to observe more detail, there is no more detail after this level. We need to define the number of bins (stratum size) that guarantees no pair of points fall into one bin.

As presented in Figure 3.11, the aggregate pyramid can be computed by a recursive SQL query. The main components processed in constructing the aggregate data pyramid

```
-- SQL aggregate pyramid template
recursive Pyramid (c_0, ..., c_{h-1},
        divs,
        b_0, ..., b_{k-1},
        a_0, ..., a_{r-1}) as (
    -- base step
    select c_0, ..., c_{h-1},
        divs,
        integer(x_0 - low_0 * divs /
                high_0 - low_0) as b_0,
            ...,
        integer(x_{k-1} - low_{k-1} * divs /
                high_{k-1} - low_{k-1}) as b_{k-1},
        base_agg_0(...) as a_0,
            ...,
        base_agg_{r-1}(...) as a_{r-1}
    from Dataset, Params
    group by c_0, ..., c_{h-1},
        divs,
        b_0, ..., b_{k-1}
    union all
    select c_0, ..., c_{h-1},
        integer(divs / 2) as sdivs,
        integer(b_0 / 2) as sb_0,
            ...,
        integer(b_{k-1} / 2) as sb_{k-1},
        ind_agg_0(...) as sa_0,
            ...,
        ind_agg_{r-1}(...) as sa_{r-1}
    from Pyramid
    where divs > 1
    group by c_0, ..., c_{h-1},
        sdivs,
        sb_0, ..., sb_{k-1}
) -- end of the recursive definition
select c_0, ..., c_{h-1},
    -1 * integer(log(2, divs)) as stratum,
    zo(b_0, ..., b_{k-1}) as zoo,
    b_0, ..., b_{k-1},
    a_0, ..., a_{r-1}
from Pyramid
```

Figure 3.11: SQL aggregate pyramid template

are illustrated in Table 3.2. In the base step, the base of the pyramid is constructed. In this step, the size of bins along all desired dimensions is determined for the base of the aggregate pyramid which defines what is the smallest bin size to hold the aggregate values. The number of bins is a power of two for all dimensions. To build the aggregate pyramid, one dimension is, or a group of dimensions are, processed to form a one-dimensional or

Dataset:	assemble a dataset of interest from the database
c_i's:	attributes with "categorical" data (c_0, …, c_{h-1})
x_i's:	the dimensions (x_0, …, x_{k-1})
m_i's:	the measures (m_0, …, m_{q-1})
Params:	parameters used for constructing the pyramid; derived over dataset and/or defined by user; *returns one row*
low_i's:	lower grid boundaries for the pyramid
$high_i$'s:	higher grid boundaries for the pyramid
divs:	how many bin-divisions along each x_i for the pyramid base; is a power of 2 (e.g., 1,024)
Pyramid:	the constructed pyramid
c_i's:	constructs a pyramid per partition of c_0, …, c_{h-1}
divs:	number of bin-divisions (indicates stratum)
b_i's:	bin number (for each dimension)
a_i's:	aggregate values
Result:	the returned aggregate pyramid
c_i's:	the constants
stratum:	stratum of the pyramid
zoo:	Z-order ordinal
b_i's:	bin number (for each dimension)
a_i's:	aggregate values

Table 3.2: Pyramid terminology.

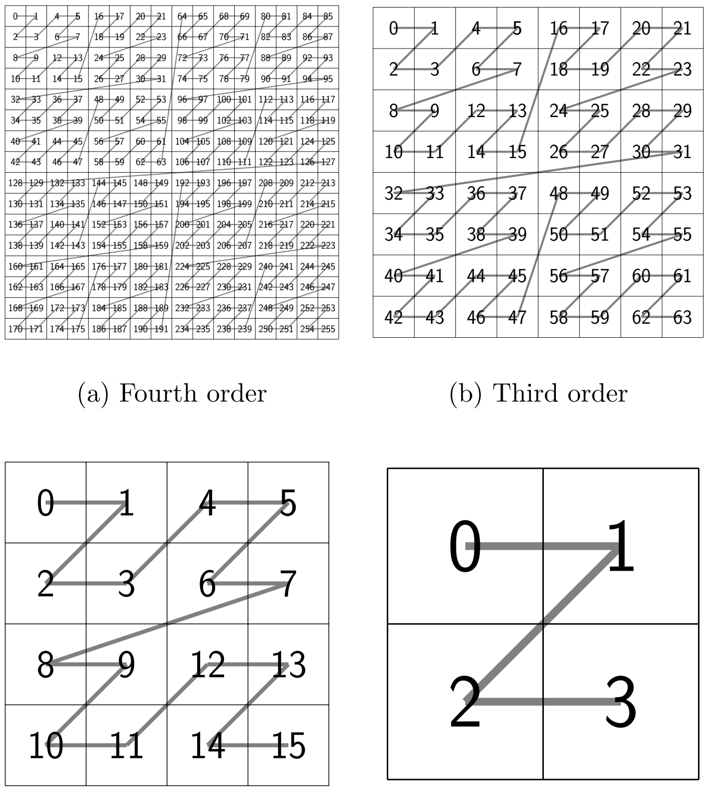
multidimensional aggregate data pyramid. The base aggregation functions *base-agg* are then applied on measures to fill the predefined bins with aggregate values. The attributes *c*s represent "constants" (this can be used to pipeline processing to build the cross products of aggregate pyramids. This is discussed in Chapter 5). In the recursive part of the SQL query then, the inductive aggregation is applied over the pre-processed and pre-aggregated dataset. In this phase, we double the size of bins along every dimension by halving the number of bins along each dimension to build one stratum higher from the previous one. The inductive aggregation functions *ind-agg* are employed on pre-aggregate values from the previous step to fill the bins with new calculated aggregate values.

3.4.3.1 Mapping Multi-dimensional data into 1D

The first step after determining the scope of requested data to be retrieved as the base dataset in the base-aggregation phase is to assign sequential Z-order key values to tuples and sort them base on their Z-order values. The next stratum is then computed by applying the re-aggregate functions over the aggregate values in the base stratum. During processing data and computing the aggregate pyramid, the inductive aggregate functions are employed over the appropriate bins of one stratum below the current stratum to compute the re-aggregate values. At the same time, we can compute the Z-order while processing and building the higher strata of the pyramid.

Figure 3.12 presents the Z-order in a $2^n \times 2^n$ two-dimensional space ($n = 4$) at four different granularities (1 to 4). The level 0 contains one cell with one key value which is zero and since for one value there does not exist a "curve", we represent the Z-order curves from level 1. At each level L ($0 \leq L \leq n$), the dimensions are divided into 2^L. These four granularities (levels) can be built in an iterative process. In the first iteration ($n = 1$), the size of the two-dimensional space is $2^1 \times 2^1$, which contains four cells (bins). The size of dimensions is doubled in every iteration. As a result, the size of the two-dimensional spaces in the second and third iterations are $2^2 \times 2^2$ and $2^3 \times 2^3$, etc., as shown in Figure 3.12.

Given a data space with granularity n, the binary representation of a data point coordinates X and Y is an n-bit binary number. The Z-order can be calculated simply by interleaving the bits of X ($x_{n-1}x_n...x_0$) and Y ($y_{n-1}y_n...y_0$) coordinates. The result



(a) Fourth order (b) Third order

(c) Second order (d) First Order

Figure 3.12: Four-level Z-order

is a $2n$-bit binary number $(y_{n-1}x_{n-1}y_nx_n...y_0x_0)$, representing the Z-order for the point $P(X, Y)$. After mapping the data points to a consecutive list of values, a one-dimensional index structure such as a *B+-tree* can be used to store and retrieve the aggregate data points at each level. The index of each data point (a bin) is a combination of the depth of the containing stratum and the Z-order of the data point in the aggregate pyramid with depth n.

$Index$: L (the level) + Z-order

	0	1	2	3	4	5	6	7
0	0	1	4	5	16	17	20	21
1	2	3	6	7	18	19	22	23
2	8	9	12	13	24	25	28	29
3	10	11	14	15	26	27	30	31
4	32	33	36	37	48	49	52	53
5	34	35	38	39	50	51	54	55
6	40	41	44	45	56	57	60	61
7	42	43	46	47	58	59	62	63

(a) Level 3

	0	1	2	3
0	0	1	4	5
1	2	3	6	7
2	8	9	12	13
3	10	11	14	15

(b) Level 2

	0	1
0	0	1
1	2	3

(c) Level 1

Figure 3.13: Z-order values of three levels

Algorithm 1 Calculating Z-order value of n dimensions

```
procedure Z_NUMBER(array dimensions[])– x,y,z,...
    mask ← 1
    z_number ← 0
    d ← dimensions.Length()
    bit_place ← 0
    while mask > 0 do
        for i from 1 to d do
            masked_bit ← dimension[i] ∧ mask
            if masked_bit > 0 then
                bit ← 1
            else
                bit ← 0
            end if
            bit ← bit ≪ [(i − 1) + (bit_place ∗ d)]
            z_number ← z_number ∨ bit;
        end for
        bit_place ← bit_place + 1
        mask ← mask ≪ 1
    end while
    return z_number
end procedure
```

Figure 3.13 shows an example of Z-order values in three granularities. Consider the point denoted by coordinates $X = 6$ and $Y = 4$ at level three with size $2^3 \times 2^3$, which binary representation of X and Y are respectively $x_3x_2x_1x0 = 110$ and $y_3y_2y_1y_0 = 100$. The 8-bit Z-order value of the point is then $y_3x_3y_2x_2x_1x_1y_0x_0 = 110100$ which corresponds to number 52. The Z-order curve is easy to order and nicely preserves spatial locality. Algorithm 1 demonstrates the process of mapping a point from a multi-dimensional space to a one dimensional data value (Z-order) by interleaving the bits of its coordinates.

3.4.3.2 Depth of the Aggregate Pyramid

One essential issue in building the aggregate pyramid is the depth of the pyramid. We need to determine how deep the pyramid needs to be computed. We consider the deepest height that we can build a data pyramid in our system. In a 64-bit system, we can consider the dimensions X and Y as big as 32 bits. As a result, the Z-order is a 64-bit number resulted from the bit interleaving of the coordinates X and Y. Consequently, the depth of the aggregate pyramid is 33 (from 32 to 0) in which the base stratum (the stratum 32) of the pyramid is a $2^{32} \times 2^{32}$ rectangular of bins and apex (the stratum 0) is a 1×1 bin. Since this base stratum is very deep, many bins could be empty at this level. In order to improve the efficiency, we need to define which level needs to be computed from the base that has a more coherent and meaningful bin size (number of bins along each dimension). We call the depth of the second stratum built from the base stratum, natural depth. This means that if the user wants to zoom in more for further detail, there is no more detail after this level. In order to find the natural depth, we need to find the smallest distance between our points, which we call delta (δ). We consider the two-dimensional closest pair problem in order to find the bin size of the natural depth; given n points in a two-dimensional space, we need to find a pair of points whose distance δ is smallest. After finding δ, it is not necessary to divide data dimensions into the bins smaller than δ. To find the smallest distance between data points, we can apply some techniques such as Metric Distance Measurement. We also define a technique to find approximately the natural depth called Super Bin Adjacency. We discuss both techniques below.

3.4.3.2.1 Metric Distance Measurement

In metric distance measurement, we find the closets distance of a pair of points to define the natural depth. In this approach, we can apply two metric techniques in order to find the smallest distance (δ);

- **Euclidean distance** Euclidean distance [81] between two points is the length of a straight path between them. In a two-dimensional space, the Euclidean distance between a pair of points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ is calculated by the Pythagorean formula:

$$\delta = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- **Chebyshev Distanc** Chebyshev Distance [81] Chebyshev Distance, conceived of by

Pafnuty Chebyshev, is a metric measurement to find the distance of a pair of points in a multi-dimensional space. In this technique, the greatest difference of two points' coordinate dimensions is considered as the distance of those points. The Chebyshev distance of a pair of points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ is calculated by the following formula:

$$\delta = \max(|x_2 - x_1|, |y_2 - y_1|)$$

Chebyshev distance metric is computationally simpler compared to Euclidean distance technique. However, this metric does not have the rotation-invariant property since in a rotation transformation, the distance between points might be changed in Chebyshev distance metric. It means that if we rotate the grid, the Chebyshev distance will differ and could be smaller than the distance we compute to build our bins at the natural depth. Euclidean distance, on the other hand, is rotation invariant but it has more complexity to find the closet pair of points. We resolve this problem by applying the Chebyshev distance approach to find the smallest distance which defines the size of the pyramid's bins. Then, we halve the bin size calculated in Chebyshev distance metric to solve the rotation variant problem of this metric. It means we build our natural depth stratum by dividing our dimensions into bins with the number of bins are two times bigger than the calculated number. In other words, we construct the aggregate pyramid from one level below the calculated depth but not lower than base stratum.

Algorithm 2 Smallest Distance Calculation using Chebyshev Metric for 2D Data

```
    procedure SMALLEST_DISTANCE(array z_order[], dimension_size S)
Require: An array of z_order values, size of dimensions in bits
        array distance[] ← NULL
        if z_order.Length() > 1 then
            $x\_mask \leftarrow 1$
            $y\_mask \leftarrow 2$
            for $i$ from 1 to $S$ do
                $x\_bit_0 \leftarrow z\_order[1] \wedge x\_mask$
                $y\_bit_0 \leftarrow z\_order[1] \wedge y\_mask$
                $x\_bit_0 \leftarrow x\_bit_0 \gg (i - 1)$
                $y\_bit_0 \leftarrow y\_bit_0 \gg i$
                $x_0 \leftarrow x_0 + x\_bit_0$
                $y_0 \leftarrow y_0 + y\_bit_0$
                $x\_mask \leftarrow x\_mask \ll 2$
                $y\_mask \leftarrow y\_mask \ll 2$
            end for
            for $i$ from 2 to z_order.Length() do
                $x\_mask \leftarrow 1$
                $y\_mask \leftarrow 2$
                for $j$ from 1 to $S$ do
                    $x\_bit \leftarrow z\_order[i] \wedge x\_mask$
                    $y\_bit \leftarrow z\_order[i] \wedge y\_mask$
                    $x\_bit \leftarrow x\_bit \gg (j - 1)$
                    $y\_bit \leftarrow y\_bit \gg j$
                    $x \leftarrow x + x\_bit$
                    $y \leftarrow y + y\_bit$
                    $x\_mask \leftarrow x\_mask \ll 2$
                    $y\_mask \leftarrow y\_mask \ll 2$
                end for
                $distance[i-1] \leftarrow greatest(x - x_0, y - y_0)$
            end for
        end if
        return smallest( for all in distance)
    end procedure
```

As mentioned before, Chebyshev Distance is a metric to define the distance between two points by finding the greatest difference along tow points' coordinates. While processing the raw data to aggregate and build the base stratum, we also calculate the Z-order and find the smallest distance between points. First, we assign a set of consecutive values to every bin in the base stratum applying Z-order space filling curve technique by interleaving the bits of points' coordinates. Next, we ascendingly sort the base stratum based on the Z-order values. Then, the Chebyshev metric is applied to find the smallest distance between two points in the dataset. In this process, we consider four preceding continuous points P_k, P_{k-1}, P_{k-2}, and P_{k-3} to be examined. We apply Chebyshev metric to find the closet pair in the group of four selected points in which the distance of a point is calculated from its three preceding points. We keep the smallest distance as δ to compare with the smallest distance of the next heading points (Algorithm 2). Each time, we can sample four points to find the smallest distance which can give us a satisfying approximation. To calculate the dimension size D_n of the natural depth;

$\delta =$ approximate smallest distance defined in the base-aggregation phase.

$X_{max} =$ greatest number in x coordinate of the base stratum

$Y_{max} =$ greatest number in y coordinate of the base stratum

$D_n = \lceil \log_2(\lceil \max(X_{max}, Y_{max})/\delta \rceil) \rceil$

After finding the smallest distance (δ) and the number of bins in each dimension, the size of the natural depth is then $2^{D_n+1} \times 2^{D_n+1}$. We build the natural depth one level below the calculated depth to guarantee the rotation invariant property. The rest of the pyramid is then constructed by applying the inductive aggregation on the aggregate windows (groups of 2×2 aggregate quadrants) from the very lower level.

3.4.3.2.2 Super Bin Adjacency

As described before, each bin is correlated to a super bin from the very next higher stratum and each super bin is associated with four sub-bins from the very lower stratum. The strata of the aggregate data pyramid are made of bins and each bin represents one single aggregate value with a unique Z-order value in the containing stratum.

To apply the Super Bin Adjacency approach in a two-dimensional space, we compare each point with its two neighbors; a succeeding point and a following point in the Z-order sequence to see if they fall into the same super bin of the one stratum higher (with lower resolution). If two examined points are contained in a same super bin of the next higher stratum, they are adjacent in the current stratum. If those points are not in the same super bin in one stratum higher, we continue examination until we find the depth at which they do fall into a same super bin. We store the associated depth of each group of three points (a current point and its two immediate neighbors) and continue to examine the rest of the points of the base stratum. In each three-point examination, if we found any associated depth greater than the previous calculated depth, we consider the new one as the natural depth.

We assume the depth of the pyramid is $n+1$ (deepest n to lowest 0). As a result, our base stratum is a $2^n \times 2^n$ two-dimensional matrix. As discussed in the previous section, Z-order is the result of the bit interleaving of X and Y. Since to represent each X and Y we need n bits, $2n$ bits are required to represent the Z-order that conveys n pairs of bits.

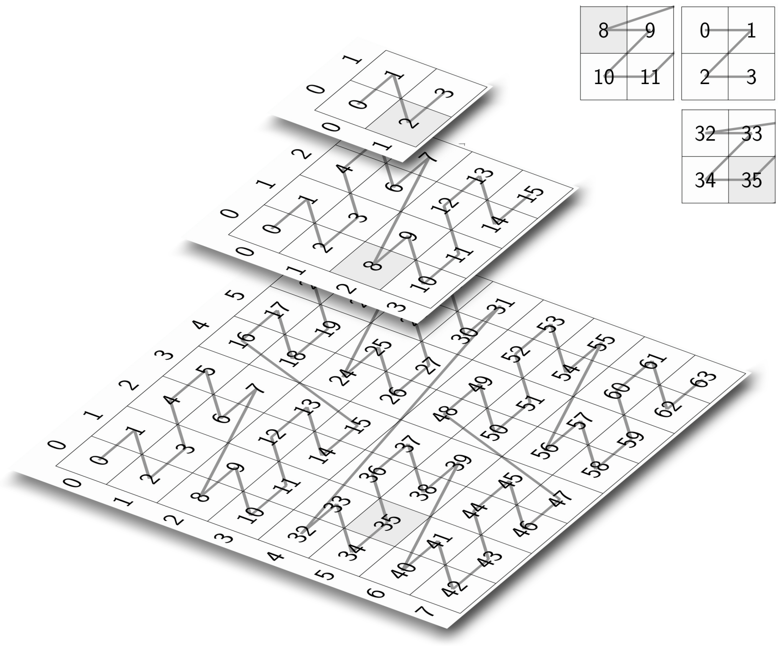


Figure 3.14: Quadrant-recursive property of Z order

The first right two bits present the place of the point on the local Z-order curve in the aggregate window (Figure 3.14). We call the curve inside an aggregate window the local curve. The rest of the bits then represent the Z-order of the super bin containing the current aggregate window in one stratum higher. The remaining bits are analyzed in the same way. The second two right bits represent the place of the correlated super bin (parent bin) on the local curve of its aggregate window. For instance, the binary representation of the point with Z-order 35 on the depth 3 of the aggregate pyramid is 100011. Since the current depth d is 3 ($d = 3$), the number of bits required to represent the Z-order is 6 bits ($2d$). The most right two bits (11) represent the number 3 specifies the place of the point on the local Z-order curve in its aggregate window which is the fourth place (place 3). The rest of the bits (1000) define the Z-order 8 of the correlated super bin on the very next higher stratum ($d = 2$) containing the initial aggregate window in one level below (stratum 3). The most right two bits in the binary representation of the bin with Z-order 8 similarly specify the place of the point $P(0, 2)$ on its local curve in its aggregate window which is the place 0. The remaining bits are then analyzed in the same way as required.

In order to find the depth at which all points are separated (so that they never fall into the same bins), we can compare the binary representation of Z-order values assigned to the points. As discussed above, in this approach, each point's Z-order is compared to its immediate neighbors' Z-order. We compare the pairs of bits from the most left side of the binary representation of the Z-order of the initial point with its preceding and following neighbors. If the first pairs of two points' Z-order are similar, we move to the next pairs from the left until we meet the pairs which are not identical. We have n pairs of bits in total in depth n, the deepest level (base stratum) where we examine all the points to find the natural depth (the deepest meaningful depth of the pyramid). A pair of bits holds a place ρ on the Z-order value numbered from 1 to n (from left to right), each representing the stratum number of a correlated depth to be compared. In comparing two Z-order values from the left most bit-pair place, when we meet two pairs of bits that are not identical, the place ρ of the pairs indicates the depth that two points are adjacent, so they also fall into separate bins in that stratum with depth number ρ. We consider this depth as the local natural depth of two points, to compare to the local natural depth of other points. To find the natural depth of a given dataset, we examine a sequence of three points (the initial points and its immediate neighbors on Z-order) retrieved from the dataset. After finding the local natural depth of the current point and its immediate neighbors, we check the next sequence of three points until all the points in the base stratum are examined. Algorithm 3 demonstrates the process of finding the natural depth of multi-dimensional data points using the Super Bin Adjacency technique.

3.4.3.2.3 Issues with the Natural Depth

In summary, to find the natural depth, we examine the four preceding points P_k, P_{k-1}, P_{k-2}, P_{k-3} in Chebyshev metric approach and three points P_k, P_{k-1}, P_{k-2} in Super Bin Adjacency. After finding the greatest depth (the local natural depth) for a set of points, we compare the new depth with previous computed local natural depth to choose the greatest one. We continue until all points of a given dataset are examined. We then build the final natural depth one level below the calculated depth (not deeper than the base stratum) in order to resolve the rotation invariant issue. In fact, in the process of finding natural depth, we examine the points to find the natural depth for every single point. It means that each point is separated from other points in its natural depth, but it has at least one immediate non-empty neighbor. Therefore, the natural depth of the pyramid which has to guarantee that all points are separated is the greatest natural depth of all the points. The natural depth of a single point can be smaller than the natural depth of the aggregate pyramid that means the point remains individual and is not merged with

Algorithm 3 Local Natural Depth Calculation using Super Bin Adjacency

```
procedure NATURAL_DEPTH($array\ z\_order[], dimension\ D, dimension\_size\ S$)
Require: An array of z_order values, the number of dimensions, size of dimensions in bits
    $depth \leftarrow 0$
    $stratum \leftarrow 0$
    $n\_bit \leftarrow S * D$
    if $z\_order.$Length() $> 1$ then
        for $i$ from 1 to $z\_order.$Length() do
            $init\_mask \leftarrow init\_mask +$ power$(2, (n\_bit - i))$
        end for
        $mask \leftarrow init\_mask$
        for $i$ from 2 to $z\_order.$Length() do
            while $mask > 0$ do
                $stratum \leftarrow stratum + 1$
                $masked\_number\_0 \leftarrow z\_order[1] \wedge mask$
                $masked\_number \leftarrow z\_order[i] \wedge mask$
                if $masked\_number\_0 \neq masked\_number$ then
                    if $stratum > depth$ then
                        $depth \leftarrow stratum$
                    end if
                    exit
                end if
                $masked\_number\_0 \leftarrow 0$
                $masked\_number \leftarrow 0$
                $mask \leftarrow mask \gg D$
            end while
            $mask \leftarrow init\_mask$;
            $stratum \leftarrow 0$
        end for
    end if
    return $depth$
end procedure
```

any other points until its own natural depth. We call bins with no immediate non-empty neighbors as islands. Islands are bins with raw or aggregate data values such that their surrounding bins in the Z-order mapping curve are empty. A point with natural depth 24, for instance, in the aggregate data pyramid with depth 29, is an island in depths between

29 to 24. Depth 24 is its natural depth meaning that the point is separated from other points but it has at least one immediate non-empty neighbor which is aggregated with its neighbor in the very next stratum, depth 23. Choosing the natural depth very deep can result many bins to be islands in the deep strata. The islands are not immediately aggregated with others points in the inductive aggregation process. As a result, they are replicated at each stratum from the base stratum until their own natural depth. This issue can cause the size of the pyramid become very large as the result of island replication. This imposes a negative impact on the proficiency and performance of constructing the aggregate pyramid.

3.4.3.3 Indexing the pyramid

The materialized pyramid provides an interactive exploration in order to visualize the requested data. We can define the pyramid as a relation in our database that a tuple T_p represents a bin B_p in the pyramid. Each tuple T consists of the corresponding stratum S (the granularity that the aggregate value is calculated), Z-order, the coordinates D (X and Y in a two-dimensional dataset), the aggregate values A, and other required parameters P:

$$B_p = T(D, S, Z, A, P)$$

In order to improve the performance of the interactive exploration on the aggregate pyramid, we index the pyramid. A proper index facilitates the query processing over our pyramid during the process of the interactive data exploration and visualization. In the process of constructing the pyramid, we assign a distinct number (Z-orders) to the bins in a correlated stratum. In fact, by assigning the Z-order values to the points, we map a two-dimensional space of each stratum into the one-dimensional data points. Therefore, the pyramid tuples (bins) in each stratum can be indexed by the *B+-tree* index structure on Z-order and the stratum number which sufficiently supports the interactive operations needed over our hierarchical data structure.

An other index structure that can be employed to index the aggregate pyramid is the quadtree. However, to support the interactive operations over the aggregate data pyramid such as zooming and panning, the *B+-tree* is an appropriate index structure since it supports range queries in an optimum and efficient way. Therefore, it is beneficial to apply the *B+-tree* index on the aggregate pyramid since this index structure is sufficient to perform range queries required for interactive operations over the pyramid.

Chapter 4

API

As datasets get ever larger, efficient interactive visual information exploration and analysis receive significant attention from the database community. Interaction is essential in data visualization and exploration, especially for the process of analyzing and visualizing large datasets where the large volume of data imposes a great challenge to the real-time interaction [82]. The aggregate pyramid improves data processing and reduces delays by pre-computing and materializing the raw data in a multiple projection and representation structure. It also provides local (horizontal) processing (panning) by querying current stratum and resolutional (vertical) processing (zooming in and out) by querying among different strata with different resolutions.

Typically, the first request issued from the user is processed to provide an overview of the entire dataset. In the data visualization API, when the user issues a query, the system evaluates the request to determine whether the visual representation of the requested data can fit the canvas. When the user has a display with limited size, the system sends a result consisting the data from the lowest resolution stratum compatible with the canvas size. The canvas is the rectangular area on the screen; data is visualized in that area. The size and location of the canvas on the screen can differ in various applications. The size of the canvas can be same as the size of the screen or smaller. When the dataset is very large, in order to scale the data to the canvas, the system applies inductive aggregation in order to create a hierarchical dataset. The aggregate data is then sent to the visualizer at a lower resolution to provide an overview of the requested data. In this approach, the user receives a quick feedback which is fast enough to support real-time interaction. To display the full resolution of data in further detail, when the scale of the requested data and display is different, the user needs to zoom and pan interactively into strata with higher resolutions.

In many applications with loose coupling, the database exports a subset of data imported to the visualization system in the client application [83,84]. The visualizer (front-end) system is then responsible to process and explore data, which itself requires data management features. In tight coupling, however, less data management is needed on the visualization side. Most requests from the user to query and explore data are processed and maintained by database management systems.

The aggregated data pyramid is an appropriate structure to be used for an efficient data visualization process. Given a query to explore a dataset, the database system materializes the data-pyramid version of the dataset, and indexes the pyramid by *stratum* and Z-order. The aggregate pyramid is then managed by the database engine during the dataset visual exploration. The aggregate pyramid provides range queries over a rectangular region by indexing the two-dimensional data points based on the Z-order that maps two-dimensional data points into one-dimensional space, and the stratum number. The Z-order value in the index keys utilizes bit-wise manipulation to facilitate the search of a subset of data in a multi-dimensional space. Visual interactive operations such as zooming and panning are supported by querying over the data pyramid at the appropriate stratum and desired range, which provides a real-time and efficient interaction. In other words, in a data visualization process, the aggregate data pyramid maintains database support in order to manage and process very large datasets in an interactive manner. The design of the aggregate data pyramid efficiently supports the following visual operations over large datasets in an interactive visualization.

4.1 Resizing

In an interactive visual exploration of a dataset, the user can change the size of the visualized data (size up or size down). In resizing, the visualizer needs to present a different resolutions of the same window of interest. The same window of interest in different strata cover different number of aggregate data points (bins). A boundary of super bins from a lower resolution stratum contains all correlated sub-bins from a higher resolution stratum.

The window of interest (query window) is the rectangular bounding box that defines the borders to retrieve and process the aggregate data from the same or different stratum.

The window of interest could be the size of the canvas or to be defined by the user as a rectangle boundary to select a specific region for more processing. In the re-sizing operation, the initial window of interest is the size of the canvas. In fact, the canvas is the boundary for retrieving and visualizing data. A query window in different strata contains a different number of data points. For instance, in a system with the aggregate pyramid of depth five (4 to 0), a query window in stratum 1 can be translated to a boundary in stratum 4.

In a size-up operation, a translated boundary window from a higher data resolution is visualized. In this operation, more detail is retrieved from the database into the memory. The user defines the requested resolution by maximizing the canvas. Associated data is then retrieved and sent to the visualizer. After having data in requested size and detail, other interactive operations such zooming and panning are available to the user. In the size-down process, on the other hand, data from the stratum with lower resolution contained in the determined window of interest is visualized. In this process, the user specifies the requested resolution by minimizing the canvas. The requested lower resolution fitting the canvas is retrieved and sent to the visualizer.

An upper bound of size-up process is the display resolution. In the case that the resolution of the deepest stratum (the base stratum) is smaller than the display, the upper bound of the size-up operation is the resolution of the base stratum, since no more data is available below this resolution. Similarly, the lower bound of the size-down operation is the available lowest resolution in the aggregate data pyramid (the pyramid apex).

4.2 Zooming

In image processing, zooming changes the number of display pixels per image pixel. In a zooming operation, the user may request more or less detail of the image. Similarly, in data visualization, zooming changes the number of display pixels per data point (bin). The zooming operation is applied by considering the center of the canvas as the pivot point to retrieve new bins with aggregate data values from the aggregate pyramid. In a re-sizing operation, the window of interest is constant and does not change during the re-sizing process while the canvas changes. In fact, in the re-sizing process, same window of interest in different resolution is represented. In the zooming operation, in contrast, the window of interest is subject to change while the size of the canvas is fixed. Every time user zooms in to observe further details from the aggregated pyramid, more data is fetched. In the zooming-in operation, the smaller area with more detail is visualized. In the zooming-out operation, however, less detail of a larger area is represented. In other words, the zooming operation produces different views with different resolutions while re-sizing process generates same view in the various size and resolution.

4.3 Panning

In panning, the resolution scale is constant. In fact, the window of interest is constant in this operation while the area to be explored and visualized changes. In this process, the user changes the view point of the visualized data within the same resolution. If the user requests different areas to be visualized by panning, the system will check the availability of data with same resolution in the memory. If requested data is not available in the memory, the system retrieves more data from the target relation at the same resolution.

4.4 Region of Interest (ROI) (Data of Interest)

A region of interest is a selected subset of visualized data within a dataset defining a high resolution area in the current stratum. In other words, an ROI is identified by the resolution and a rectangle defining the window of interest, including a smaller area. Indeed, the region-of-interest is initially same as the canvas. In a region of interest selection, the user specifies a window of interest in a current resolution (strata) by defining a rectangular window. The corresponding data contained in that region is then retrieved from a higher resolution stratum. In this process, the user selects a smaller rectangular area to observe more detail in that specific region. The data related to the requested region is retrieved from a deeper stratum of the aggregated pyramid and visualized in a resolution compatible with the canvas resolution.

4.5 Implementation of Interactive Operations

The hierarchical structure of the aggregate pyramid, indexed on *stratum* and Z-order in the *B+-tree* structure, efficiently supports interactive operations to explore and visualize data at different resolutions of various windows of interest. The interactive operations can

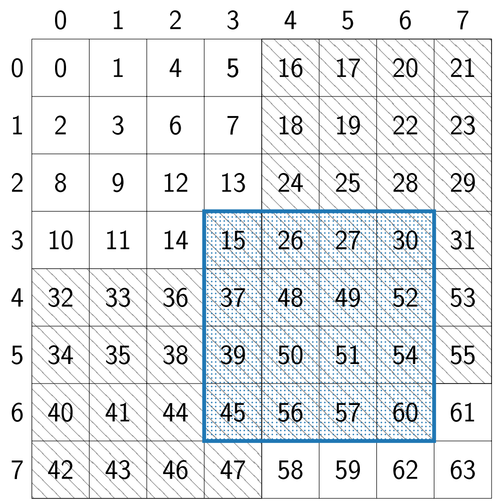
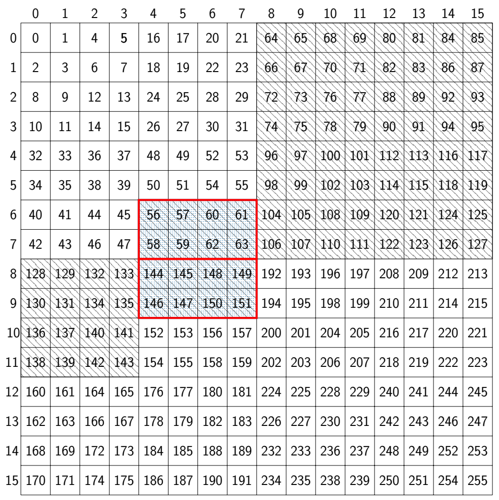


Figure 4.1: An example of a window of interest

be efficiently implemented based on the quadrant-recursive property [30] of Z-order. We define the window of interest (query window) as a square area for which its dimensions are a power of two (Again, without loss of generality, assume the same power of two along each dimension, x and y.). A window of interest (query window) may not include a contiguous range of Z-order values. As a simple solution, we can just fetch all points between minimum and maximum Z-order values in the query window (window of interest). The window of interest in Figure 4.1, specified in the blue area ($2^2 \times 2^2$), includes 16 points in the stratum 3. To have the points within the query window, we can fetch all points with Z-order values between 15 and 60; the number of retrieved tuples is three times bigger than the absolute number of tuples needed to be fetched for the initial window of interest. Figure 4.2 represents another example of a same size window of interest in stratum 4. Employing same approach to fetch the tuples contained in this window, we



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	4	5	16	17	20	21	64	65	68	69	80	81	84	85
1	2	3	6	7	18	19	22	23	66	67	70	71	82	83	86	87
2	8	9	12	13	24	25	28	29	72	73	76	77	88	89	92	93
3	10	11	14	15	26	27	30	31	74	75	78	79	90	91	94	95
4	32	33	36	37	48	49	52	53	96	97	100	101	112	113	116	117
5	34	35	38	39	50	51	54	55	98	99	102	103	114	115	118	119
6	40	41	44	45	56	57	60	61	104	105	108	109	120	121	124	125
7	42	43	46	47	58	59	62	63	106	107	110	111	122	123	126	127
8	128	129	132	133	144	145	148	149	192	193	196	197	208	209	212	213
9	130	131	134	135	146	147	150	151	194	195	198	199	210	211	214	215
10	136	137	140	141	152	153	156	157	200	201	204	205	216	217	220	221
11	138	139	142	143	154	155	158	159	202	203	206	207	218	219	222	223
12	160	161	164	165	176	177	180	181	224	225	228	229	240	241	244	245
13	162	163	166	167	178	179	182	183	226	227	230	231	242	243	246	247
14	168	169	172	173	184	185	188	189	232	233	236	237	248	249	252	253
15	170	171	174	175	186	187	190	191	234	235	238	239	250	251	254	255

Figure 4.2: An example of a window of interest

need to retrieved all the tuples with Z-order values between 56 and 151 which needs tuples six times more than the tuples included in the initial query window to be fetched. Looking at the window of interest in Figure 4.2, there exist two continuous ranges; 56 to 63 and 144 to 151. Therefore, we can query the stratum 4 of the aggregate pyramid to retrieve the tuples inside the window of interest over these two rectangular areas specified by red borders.

Quadrant-recursive property [30] of Z-order determines that each quadrant (a cell) can be decomposed into sub-quadrants with consecutive Z-order key values. The Z-order 2 in stratum 2 is decomposed to four sub-quadrants (2×2) in stratum 3 and sixteen sub-sub-quadrants $(2^4 \times 2^4)$ in stratum 4. To query the correlated stratum in order to fetch the tuples inside the window of interest, we can divide the query window to continuous Z-order intervals employing the quadrant-recursive property of the Z-order. A window

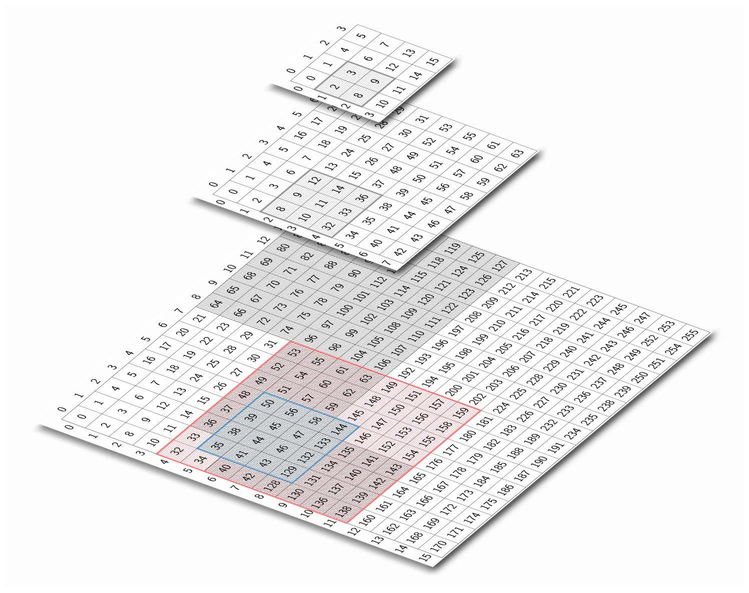


Figure 4.3: An example of a window of interest

of interest could consist of one or more contiguous ranges. To fetch the tuples of the query window, we query the corresponding stratum over at most four Z-order intervals. A window of interest can be covered by one or more super bins from higher strata called covering super bins. In the best scenario, the correlated covering super bins are exactly mapped onto the window of interest. However, it is most likely that the associated super bins overlay an area larger than the query window. The window of interest in Figure 4.2 can be covered by four super bins from one stratum higher (stratum 3). These super bins can be examined to determine the probes. In this example, the query can be executed via two probes. In Figure 4.2, four super bins are precisely overlap the initial window of interest. As a result, the exact number of tuples included in the query window are fetched.

Figure 4.3 illustrates a query window with size $2^2 \times 2^2$ in stratum 4 ($2^4 \times 2^4$). In the simple apprach, for querying the window of interest to fetch tuples between minimum and maximum Z-order values (35 and 144) of the initial window of interest, the number of tuples required to be retrieved is approximately six times more than the number of initial requested tuples. However, in the super bin coverage (covering) approach, the number of tuples needed to be fetched is at most four times bigger than the number of tuples in the initial query window. Applying the super bin covering approach on the example shown in Figure 4.3, nine super bins from one stratum higher (stratum 3) include the initial window of interest. Since, we want at most four super bins to cover the initial query window, we move one more stratum up where four super bins overlay the initial query window. The red rectangular box indicates the area covered by super bins from stratum 2. The new query window is larger and has the size four times bigger than the size of the initial query window.

In the super bin coverage approach, the final query result is at most four times bigger than the number of tuples in the initial query window. For a given stratum with size $2^n \times 2^n$, assume that the initial query window w has the size $2^k \times 2^k$ ($k \leq n$). In the best case, the query window w covers at most four contiguous ranges of Z-order values. In this case, the initial query window covers all quadrants of associated bins from the higher stratum. As a result, the number of retrieved tuples is exactly the same as the number of bins in the initial query window (see 4.2). Now, consider the case that the initial query window w partially covers the quadrants of the correlated super bins from the next higher stratum and does not include at most four continuous ranges of Z-order values (see 4.3). Therefore, a larger area has to be examined to fetch the selected bins in w from the initial resolution. In this technique, we are interested to find a query window that covers all quadrants of the correlated super bins and covers at most four Z-order value ranges. The query window is a square area with its dimensions as a power of two. To select a larger area covering the initial query window, we want to examine the smallest query window w', which covers the initial query window w. Since the size of initial query window is $2^k \times 2^k$, the smallest window covering w has the size $2^{k+1} \times 2^{k+1}$ ($k+1 \leq n$) that covers all quadrants of its associated super bins from the next higher stratum. Therefore, the smallest window w' larger than the initial query window w has the size $2^{k+1} \times 2^{k+1}$ ($4 \times (2^k \times 2^k)$) which is four times bigger than the initial query window.

In the super bin coverage (covering) technique, we define in what stratum there exist at most four adjacent super bins that cover the window of interest. To find the associated strata and super bins, we employ both super bin adjacency (Chapter 3, Section 3.4.3.2.2) and Chebyshev distance metric [81] techniques. In this technique, we just need to examine the minimum and maximum Z-order values of the query window (35 and 144). First, we apply super bin adjacency approach to find the stratum where these two data points are adjacent. In the Z mapping curve, each level of ordering (n to 1), can be divided into four regions, northwest, northeast, southwest, and southeast numbered from 0 to 3 which are square areas with $\frac{1}{4}$ the size of the corresponding ordering level. In the level 4 of Z ordering, for instance, the regions respectively include the bins with Z-order values from

0 to 63, 64 to 127, 128 to 191, and 192 to 255. If two bins fall into different regions, they are never included with the same super bin until the level 0 where there is only a single bin. We call these regions *exclusive* regions.

We examine the starting and ending (the most northwest and southeast) bins' Z-order values by applying super bin adjacency. Since, if two bins located in two different (exclusive) regions never fall into a same super bin in intermediate strata, after finding the stratum where these two bins are adjacent using super bin adjacency technique, we employ Chebyshev metric to check the distance between the determined covering super bins. If the distance is 1, we move one stratum down to check whether these two points are still neighbors in a higher resolution level applying the distance metric. We continue to examine lower strata until the distance between two associated covering super bins is bigger than one. Ultimately, the lowest stratum with distance one between two examined covering super bins is considered as the resolution containing covering super bins. Two examined bins in the corresponding stratum can have following states;

- Two determined covering super bins are diagonal neighbors; this means that we have four super bins covering the window of interest. These bins could have either consecutive or distinct Z-order values. There exist some cases that two pair of bins (northwest/northeast and southwest/southeast) have contiguous numbers. In this case, there are two probes overlapping the query window.

- Two determined covering super bins are side neighbors; in this situation, there only exist two super bins covering the initial window of interest. If two super bins are horizontal neighbors which occupy the northwest and northeast spots in the covering super bin window (of at most four bins) and the difference of their Z-order values is one (they are contiguous on Z ordering), the window of interest can be queried over one single probe. If two covering super bins are vertical neighbors (northwest and southwest bins), two probes are required to query the correlated stratum.

After finding the associated stratum and Z-order values of two covering super bins, they are examined to see whether they are diagonal or side neighbors. Then, the required continuous Z-order ranges can be defined to query the requested data which include no more than four probes. Algorithm 4 demonstrates the process of finding covering bounding box of at most four super bins which overlays the window of interest in the requested resolution. The bins in the four corners of the window of interest and the current stratum are required in this procedure. Then the boundary window to fetch the data at requested resolution can be determined. The resolution specifies how many strata lower or higher the data has to be fetched. In the panning process, the resolution ratio is 0 since the same resolution data from different areas is required to be visualized. In the zooming and sizing operations, however, the desired resolution differs from the current one. In zoom-in and size-up processes, we look at higher resolution strata for further detail. Therefore, the resolution ratio is a positive number not bigger than the resolution of the base stratum. In zoom-out and size-down procedures, the resolution ratio is a negative number and higher strata with lower resolutions are needed to be queried. When the user asks for a lower resolution (higher strata), the same technique is applied. The

Algorithm 4 Covering Super Bins Determination for 2D Data

```
    procedure COVERING_QUERY_WINDOW($array\ z\_order[], stratum$)
Require: An array of z_order values, current stratum
2:      $depth \leftarrow 1$
        $is\_neighbor \leftarrow 1$
4:      $n\_bit \leftarrow 2 * stratum$
        $depth \leftarrow$ NATURAL_DEPTH($array[z\_order[1], z\_order[4]], 2, stratum$)
6:      while $is\_neighbor = 1$ do
            $nw\_masked\_super\_bin \leftarrow z\_order[1]$
8:          $se\_masked\_super\_bin \leftarrow z\_order[4]$
            $super\_mask \leftarrow 0$
10:         for $i$ from 1 to $(depth * 2)$ do
                $super\_mask \leftarrow super\_mask + floor(power(2, (n\_bit - i)))$
12:         end for
            $nw\_masked\_super\_bin \leftarrow nw\_masked\_super\_bin \wedge super\_mask$
14:         $se\_masked\_super\_bin \leftarrow se\_masked\_super\_bin \wedge super\_mask$
            $nw\_masked\_super\_bin \leftarrow nw\_masked\_super\_bin \gg (n\_bit - (depth * 2))$
16:         $se\_masked\_super\_bin \leftarrow se\_masked\_super\_bin \gg (n\_bit - (depth * 2))$
            $is\_neighbor \leftarrow$
18:         SMALLEST_DISTANCE($array[nw\_masked\_super\_bin, se\_masked\_super\_bin], depth$)
            if $is\_neighbor = 1$ then
20:             $\boldsymbol{nw\_super\_bin} \leftarrow nw\_masked\_super\_bin$
                $\boldsymbol{se\_super\_bin} \leftarrow se\_masked\_super\_bin$
22:             $adjacency\_mask \leftarrow super\_mask$
                $adjacency\_depth \leftarrow depth$
24:         end if
            $depth \leftarrow depth + 1$
26:     end while
        $\boldsymbol{ne\_super\_bin} \leftarrow (z\_order[2] \wedge adjacency\_mask) \gg (n\_bit - (adjacency\_depth * 2))$

28:     $\boldsymbol{sw\_super\_bin} \leftarrow (z\_order[3] \wedge adjacency\_mask) \gg (n\_bit - (adjacency\_depth * 2))$

    end procedure
```

covering bounding box contained at most four super bins is determined. If the stratum of covering super bins is higher than the requested resolution, the covering window is mapped onto the desired stratum. If the resolution of determined covering window is higher than the requested resolution, the covering bounding box is then defined in the desired resolution of a lower stratum.

In the procedure of finding the covering bounding box to fetch more detail, as we move up in the hierarchy, the covering super bins cover larger areas in the desired resolution. To optimize the process, we need to find the covering super bins in the lowest possible stratum, which covers the smallest area while including the initial window of interest. Hence, fewer tuples are required to be fetch to response the user's query. To find the covering query window in Algorithm 4, first, function *NATURAL_DEPTH* (using Super Bin Adjacency technique) is called to find the stratum where the super bins containing the two bins with minimum and maximum Z-order values (the most northwest and most southeast bins) are adjacent. Next, the Z-order of the covering super bins in this depth (stratum) is calculated (Algorithm 4, lines 7 to 16). Since the examined bins could be located in two different (exclusive) regions, the depth they are considered adjacent could be very low (a very high stratum); the covering super bins from this depth overlay the very large area at the desired resolution. We examine the covering super bins in the calculated depth to see whether there is a lower stratum that still implies the neighborhood of the covering super bins. To find a lower stratum, we examine the two determined covering super bins to check the distance between them employing Chebyshev distance metric. In the calculated depth in line 5, the distance of these two super bins is definitely one since they are adjacent at this depth. We determine the distance between two covering super bins by calling function *SMALLEST_DISTANCE* (see lines 18) in one stratum below (lower stratum). If the distance between two associated covering super bins is still one, we move down to one more stratum low and repeat the process until we meet a stratum that the covering super bins in its underneath stratum are not adjacent anymore. When we find the lowest stratum with adjacent covering super bins, we calculate the other two associated super bins if they exist. We define the northeast and southwest covering super bins of the determined depth from initial bins in the window of interest (lines 27 and 28).

When all covering super bins of the associated depth are calculated, the Z-order value ranges are determined. First, the neighboring status of two ultimately calculated covering super bins is defined to see whether they are diagonal or side neighbors. If they are diagonal nearby bins, there are four different Z-order values calculated by Algorithm 4. Otherwise, they are side neighbors meaning that there is at most two probes overlapping the window of interest. They are then checked to see whether they are horizontal or vertical neighbors to determine if there is a single probe or two.

Chapter 5

Cubed Pyramid

5.1 Definition of Cubed Pyramids

In previous chapters, we discussed how to build and query (the API) an aggregate pyramid. We introduced the aggregate pyramid for a two dimensions, with this generalizing in a straightforward way to higher dimensions. A one-dimensional or multi-dimensional aggregate pyramid can be constructed by applying *inductive aggregation* over the desired dimensions. In a one-dimensional aggregate data pyramid, the *inductive aggregation* is employed over a single dimension, say *time*. After constructing the base and assigning an order of consecutive numbers to each tuple/bin ($from\ 0\ to\ n-1$) for the base stratum of n bins, *inductive aggregation* can be applied over each group of 2^d (d is the number of dimensions) adjacent points (aggregate window) in that linear order; the aggregate window is an area containing 2^d adjacent data points (bins) on the linear order where d is the number of dimensions. In a two-dimensional aggregate pyramid for instance, the aggregate window is a rectangular area and the size of the aggregate window is 4 (2^2 or

2×2). Similarly, the size of the aggregate window in a three-dimensional pyramid is 8 (2^3 or $2 \times 2 \times 2$). Let us call a single aggregate pyramid an *axis*, which can be either a one-dimensional, two-dimensional,or a higher-dimensional aggregate pyramid.

We introduce the concept of the cubed pyramids as a generalization to our inductive-aggregate pyramids, which are the cross products of two or more aggregate pyramids (axes). This generalization, in fact, generalizes and unifies aggregate pyramids as we have introduced them and data cubes from the database literature. A challenge is then defining multi-pyramid datasets (more than one multi-resolution dataset) as a result of the *inductive aggregation*. Consider a dataset with dimension attributes of time (1D) and of latitude and longitude (2D) and a measure of temperature. a visual presentation model of this dataset could provide a visualization of a temperature map while allowing the user to view the temperature data of the map in different times (time intervals). The user could view the temperature map for a given day, week, or year, without the necessity of sending more requests to the database engine and waiting for a response. To support this, we must query over a one-dimensional aggregate pyramid on dimension *time*. On the other hand, the user may want to observe the temperature map for a specific time interval in various resolutions of dimensions x and y. Hence, we also need to explore a two-dimensional aggregate pyramid on x and y dimensions (longitude and latitude). Building a single three-dimensional aggregate pyramid (axis) on x, y, and *time* is not sufficient since it does not provide the exploration on hierarchies of *time* (t) and (x, y) independently. And to build a data cube on x, y, and t is wasteful, as

x t y aggregate always together. To have freedom in exploring the data along different axes — (x, y) and (t) — independently, we propose the cubed pyramid $d_0 \times \ldots \times d_a$ ($\sum_{i=0}^{a-1} d_i = d$ where d is the total number of dimensions) where a is the number of axes and d_i defines the number of dimensions of axis i in the cubed pyramid. The cubed pyramid $Pyramid(time) \times Pyramid(x, y)$ provides a multi-resolution structure where we can observe a specific slice (fixed resolution of x and y) in different time intervals. It means that while the resolution of time is changed, the resolution of x and y is fixed. Likewise, the resolution of time can be constant while we observe various resolutions of one axis $Pyramid(x, y)$. Or the resolution of both axes can be changed to any desired level.

5.2 Cubed Aggregate Pyramid Implementation

To build a cubed aggregate pyramid, we first process the raw data and apply a multi-dimensional ordering (Z order) to assign consecutive numbers to a set of dimensions in order to map d-dimensional data points to the one-dimensional space. To construct a $1D \times 2D$ cubed pyramid of two axes $Pyramid(time)$ and $Pyramid(x, y)$, for instance, a sequence of positive numbers from 0 to $n-1$ for a base of n bins is assigned to $time$, which is processed as a single dimension to create a one-dimensional axis. For axes with higher dimension ($d > 1$) — a two-dimensional axis of x and y, for example – we employ Z ordering to map the higher-dimensional space to one-dimensional data points in a linear ordering. The next step is to define the natural depth of the cubed aggregate pyramid.

It is possible to find the natural depth of each axis independently and build the axes of cubed aggregate pyramid at different depths. In this approach, we are interested in finding a natural depth which is meaningful and comprehensive for all dimensions. We look for smallest bin size that makes a data point with d dimensions separate from other points while it has at least one immediate non-empty neighbor bin. In fact, the depth including none overlapping bins with nearby non-empty neighbors is desired. By having a proper depth, we expect a satisfying reduction in building the higher strata of the cubed aggregate pyramid as a result of inductive aggregation.

To build the cubed pyramid of $time \times 2D$ $(Pyramid(time) \times Pyramid(x, y))$ of Brightkite Check-ins dataset [85], after finding the natural depth of $time$ and (x, y) dimensions, we apply inductive aggregation on the $time$ dimension while the resolution of other axis, dimensions x and y, remains constant. Next, we inductively and independently aggregate data over x, y axis — the x and y dimensions — where $time$ is constant. Similarly, we constructed the cubed aggregate pyramid of the NYC Taxi Trips dataset [86] with three axes, $Pyramid(time) \times Pyramid(x, y) \times Pyramid(u, v)$, where first axis is the one-dimensional aggregate pyramid on $time$, and the second and third axes are, respectively, two-dimensional aggregate pyramids on source (x, y) and destination (u, v). In a similar process, the inductive aggregation is independently applied on dimension $time$, then on x and y, and ultimately on the u, v axis (the u and v dimensions).

Every time we process the dimensions and attributes of one axes, the other axes' dimensions and attributes are held constant. Figure 5.1 represents the re-aggregation

```
-- SQL Cubed Pyramid template
with recursive P_Cube0 (ax0_d0, ..., ax0_dh-1,
        ax1_d0, ..., ax1_dk-1,
        ...,
        axn-1_d0, ..., axn-1_dr-1,
        ax0_stratum, ax0_zo,
        ax1_stratum, ax1_zo,
        ...,
        axn-1_stratum, axn-1_zo,
        a0, ..., aj-1) as (
    -- retrieve all pre-processed, pre-aggregated, and raw data from the base
    select ax0_d0, ..., ax0_dh-1,
        ax1_d0, ..., ax1_dk-1,
        ...,
        axn-1_d0, ..., axn-1_dr-1,
        ax0_stratum, ax0_zo,
        ax1_stratum, ax1_zo,
        ...,
        axn-1_stratum, axn-1_zo,
        a0, ..., aj-1
    from theBase
    union all
    select integer(ax0_d0 / 2) as nax0_d0, ..., integer(ax0_dh-1 /2) as nax0_dh-1,
        ax1_d0, ..., ax1_dk-1,
        ...,
        axn-1_d0, ..., axn-1_dr-1,
        (ax0_stratum - 1) as nax0_stratum,
        Z_NUMBER(integer(ax0_d0 / 2), ..., integer(ax0_dh-1 /2)) as nax0_zo,
        ax1_stratum, ax1_zo,
        ...,
        axn-1_stratum, axn-1_zo,
        ind_agg0(...) as na0, ..., ind_aggj-1(...) as naj-1
    from P_Cube0
    where ax0_stratum > 0
    group by nax0_d0, ..., nax0_dh-1,
        ax1_d0, ..., ax1_dk-1,
        ...,
        axn-1_d0, ..., axn-1_dr-1,
        nax0_stratum, nax0_zo,
        ax1_stratum, ax1_zo,
        ...,
        axn-1_stratum, axn-1_zo
) -- end of the recursive definition
select *
from P_Cube0
```

Figure 5.1: SQL cubed pyramid template to build from the base

phase where *inductive aggregation* is applied on the base built in the base-aggregation phase to create a first axes of the cubed pyramid. In the base-aggregation phase, the Z-order is assigned to data points while raw data is pre-processed and pre-aggregated. Then the natural depth is defined to form the base of the cubed aggregate pyramid.

In building the cubed pyramid, each time, inductive aggregation is applied on the

```
-- SQL Cubed Pyramid template
with recursive P_Cubed$_i$ (ax$_0$_d$_0$, ..., ax$_0$_d$_{h-1}$,
        ax$_1$_d$_0$, ..., ax$_1$_d$_{k-1}$,
        ...,
        ax$_{n-1}$_d$_0$, ..., ax$_{n-1}$_d$_{r-1}$,
        ax$_0$_stratum, ax$_0$_zo,
        ax$_1$_stratum, ax$_1$_zo,
        ...,
        ax$_{n-1}$_stratum, ax$_{n-1}$_zo,
        a$_0$, ..., a$_{j-1}$) as (
    -- retrieve all pre-processed, pre-aggregated, and raw data from the base
    select ax$_0$_d$_0$, ..., ax$_0$_d$_{h-1}$,
        ax$_1$_d$_0$, ..., ax$_1$_d$_{k-1}$,
        ...,
        ax$_{n-1}$_d$_0$, ..., ax$_{n-1}$_d$_{r-1}$,
        ax$_0$_stratum, ax$_0$_zo,
        ax$_1$_stratum, ax$_1$_zo,
        ...,
        ax$_{n-1}$_stratum, ax$_{n-1}$_zo,
        a$_0$, ..., a$_{j-1}$
    from Axis$_i$
    union all
    select ax$_0$_d$_0$, ..., ax$_0$_d$_{h-1}$,
        integer(ax$_i$_d$_0$ / 2) as nax$_i$_d$_0$, ..., integer(ax$_i$_d$_{k-1}$ / 2) as nax$_i$_d$_{k-1}$,
        ...,
        ax$_{n-1}$_d$_0$, ..., ax$_{n-1}$_d$_{r-1}$,
        ax$_0$_stratum, ax$_0$_zo
        (ax$_i$_stratum - 1) as nax$_i$_stratum,
        Z_NUMBER(integer(ax$_i$_d$_0$ / 2), ..., integer(ax$_i$_d$_{h-1}$ /2)) as nax$_i$_zo,
        ...,
        ax$_{n-1}$_stratum, ax$_{n-1}$_zo,
        ind_agg$_0$(...) as na$_0$, ..., ind_agg$_{j-1}$(...) as na$_{j-1}$
    from P_Cube$_{i-1}$
    where ax$_i$_stratum > 0
    group by ax$_0$_d$_0$, ..., ax$_0$_d$_{h-1}$,
        nax$_i$_d$_0$, ..., nax$_i$_d$_{k-1}$,
        ...,
        ax$_{n-1}$_d$_0$, ..., ax$_{n-1}$_d$_{r-1}$,
        ax$_0$_stratum, ax$_0$_zo,
        nax$_i$_stratum, nax$_i$_zo,
        ...,
        ax$_{n-1}$_stratum, ax$_{n-1}$_zo
) -- end of the recursive definition
select *
from P_Cube$_i$
```

Figure 5.2: SQL cubed pyramid template from previous built axes

dimensions and attributes of one axis. The "constant" attributes and dimensions of other axes in the process of building an intermediate product of the cubed pyramid are the constants $c_0 \ldots c_{h-1}$ illustrated in Chapter 3, Figure 3.11. In the re-aggregation phase, first, inductive aggregation is applied on one axis' dimensions and attributes over the base of the cubed pyramid (Figure 5.1). In this phase, the different resolution of

$axis_0$ is built while the resolution of other axes' dimensions and attributes are constant. The intermediate product P_Cube_i are created from the previous version P_Cube_{i-1} for a axes ($axis_0 \ldots axis_{a-1}$) (see Figure 5.2). Cubed pyramids provide the independent exploration of multiple resolutions of each axis. One axis in the cubed pyramid of NYC Taxi Trips dataset [86], for instance, can be explored to observe various time resolutions of this dataset. For a specific time (time interval) and specific destination resolution of u and v, the resolution of the source (x, y) can be changed. A low resolution area of a set of points as source data points can be observed with a high resolution destination data points or vice versa.

5.3 Cubed Aggregate Pyramids and OLAP Roll-up Data Cubes

Cubed aggregate pyramids can be defined as partitioned data cubes where dimensions are partitioned into predefined groups. The aggregation is then performed on these groups of dimensions. The advantage of cubed aggregate pyramids is that the aggregation is calculated only on desired set of predefined group of dimensions, not all. Cubed pyramids, in addition, provide the data hierarchy over specified dimensions, axes. An axis, in fact, is an aggregate data pyramid with a hierarchy over one group of dimensions. Indeed, cubed pyramids are generalization of roll-up cubes in order to maintain spatial aggregation over dimensions. They have both concept of data cubes and roll-up operation combined to create data hierarchies to support an interactive data visualization and exploration.

Data cubes [7] are database relations constructed by aggregating data across all combination of d dimensions. In fact, it is the d-dimensional generalization of the operator *group-by*. To construct a cube, the group-by is computed on all possible combinations of d dimensions of a dataset. It can provide efficient exploration and analysis of data for decision making, knowledge discovery, and finding patterns and anomalies; but it can also be overly large, and offer meaningless aggregation for an application that does not need certain dimensions to be aggregated independently. Since, in data cubes, all possible combination of dimensions are aggregated, the size of the data cubes grows very fast when increasing the number of dimensions. Data cubes can be pre-computed and constructed in advanced so they can be used as a source for data exploration and visualization. In the example of cubed aggregate pyramids on $time$, x, and y, the data cube can be constructed over these three dimensions. However, aggregating over all combinations of dimensions is not needed. Moreover, data cubes solely do not maintain the exploration of the dimensions' hierarchies over independent groups of axes.

There exist some OLAP operations that enables a system to drill-up or drill-down analysis on a data cube. Roll-up operation performs aggregation either by aggregating data along all hierarchy levels of a single taxonomic dimension such as time or by reducing the number of dimensions. The concept hierarchy of time, for instance, can be defined as $hour$, day, $month$, $quarter$, and $year$. In the roll-up operation, to create the time hierarchy of data, the data is first aggregated on $hour$, then it is rolled up to day and then $month$, and finally it is summarized to $quarter$ and $year$. The roll-up can be define as a process of increasing the level of aggregation from more detailed data to less detailed values.

In data visualization and analysis, data cubes are prominent structures to materialize data by aggregating over different dimensions along with the roll-up operation to form the hierarchy over one or more dimensions. However, constructing a data cube is costly for higher dimensions. The size of a cube grows exponentially by increasing the number of dimensions d (2^d different combinations of dimensions). And since the cube itself does not provide dimensional hierarchy, the roll-up operation has to be performed to create the different data granularity over the concept hierarchy of one dimension; for instance, *time* (hour, day, month, quarter, and year) or *location* (neighborhood, city, province, and country). Roll-up cubes provide both aggregation and dimensions hierarchy.

In order to make OLAP cubes' construction and storage more efficient, various work has been done. ImMens [15], for instance, use the notion of partial data cubes to decrease the memory storage needed for cubes to support an interactive data visualization. Instead of constructing a single cube with d dimensions, they build several smaller data cubes with d_i dimensions ($d_i \leq 4$ and $\sum_{i=0}^{d-1} d_i = d$). To provide a higher level of aggregation, ImMens then perform roll-up operation on the pre-computed data cubes. Lins et al. [17] propose algorithms to construct and query *nanocubes*, reduced size cubes that are small enough to fit in the main memory to maintain real-time and interactive exploratory visualization. Carlos Scheidegger [87] reviews recent main techniques to visualize large-scale data sets. He analyzes techniques employed in three different systems, including $ImMens$ and *nanocubes* . Sismanis et al. [88] propose a compact and clustered structure (Dwarf) to construct, index, and query data cubes. However, performing OLAP operations such as roll-up and drill-down over dimension hierarchies are expensive, since they need to be handled externally. Later, they extended their compressed architecture to maintain roll-up data cubes, cubes with hierarchical dimensions.

Roll-up cubes are well known structures in data visualization systems. However, standard OLAP does not provide full support of spatial operation over data cubes such as spatial aggregation. Recently, some commercial technologies have been proposed to develop spatial OLAP (SOLAP) applications supporting spatial OLAP operations. SOLAP (spatial on-line analytical processing), introduced and defined by Bedrad [89] as a new type of tool to maintain a platform to support visual exploration and analysis of multidimensional, spatio-temporal data along with multilevel aggregation. Rivest et al. in their survey [90], present a new category of features, an extension of OLAP tools, to efficiently explore and analyze spatio-temporal data.

Matias et al. [91] present a category of OLAP tools to explore sptail data (SOLAP). One essential requirement in SOLAP is rolling up through spatial hierarchies. Matias et al. [91] classify a spatial hierarchy in three main categories: full semantic (a concept hierarchy of a dimension contains only alphabetic attributes, such as neighborhood, city, province, country); hybrid (some attributes of a concept hierarchy of a dimension are alphabetic and some are geometrics such as location coordinates as $latitude, longitude$ or country); and full geometric (all attributes in a concept hierarchy are geometric). Spatial roll-up can be defined and performed on a cube to create a spatial hierarchy over one or more dimensions [92–94]. Sampaio et al. [93] propose a spatial multidimensional model which is an integration of data warehouse model (DW) and the geometry object model (OGB). Their proposed model maintains the extensions of OLAP operations in their spatial model (spatial roll-up and drill-down). They define spatial roll-up as an operation that creates geometric aggregation values from most detailed values to the least detailed level.

We are interested in spatial aggregation over a dimension or a set of dimensions (our axes). In inductive aggregation, we double the size of a bin along each dimension to create a higher level aggregation with less detail. In cubed aggregate pyramids, we can choose what sets of dimensions we want to inductively aggregate. As a result, we avoid to compute the aggregate values over dimensions or sets of dimensions that we do not require to process. Cubed pyramids can be considered as the generalized form of (S)OLAP data cubes. However, in contrast to cubed pyramids, in data cubes all possible combinations of dimensions or attributes are required to be processed and aggregated. The roll-up operation can be then performed on a data cube to create dimension hierarchies. Let d be the total number of dimensions that we want to roll-up over their concept hierarchies, l be the number of hierarchical levels, and $c_{i,j}$ be the cardinality of dimension i ($0 \leq i \leq d$) in level j ($0 \leq j \leq l$). Therefore, the size (S) of the corresponding roll-up data cube with symbolic roll-up is

$$S = \prod_{i=0}^{d-1} \sum_{j=0}^{l-1} c_{i,j}$$

Assume that, for all dimensions hierarchies, each time we aggregate two values to create a higher aggregation level. For instance, assume there exist exactly two stores in each neighborhood, two neighborhoods for each city, two cities per state, and two states per country. Therefore, to roll-up the concept hierarchy of dimension *location*, each time we aggregate two values to form the next higher level of aggregation. Indeed, each time the size of computed aggregation level is half of the size of previous level with a smaller sub-category. With all dimensions having same number of bins b ($b = 2^n$), the size of the roll-up data cube is

$$S \doteq (2b)^d$$

Let's $B = b^d$ be the size of the base dataset (data cube) required in constructing the hierarchy. Therefore, we have

$$S \doteq 2^d B$$

In the next section, we define the size of cubed aggregate pyramids in more detail.

5.4 Cubed Aggregate Pyramids Cardinalities

In Section 3.4.2.2, the size of a two-dimensional aggregate pyramid was defined. After building the base of the aggregate pyramid, stratum i is constructed from one stratum below, stratum $i + 1$. The size of the base stratum for two dimensions ($d = 2$) and the natural depth n is $2^n \times 2^n$. The size of stratum i, is $2^i \times 2^i$. Each time we double the size of bins along each dimension. Therefore, the number of bins in each dimension in stratum i is half of the number of bins in stratum $i+1$ and size of the stratum i is a quarter size of stratum $i+1$. The following formula shows the relation of stratum i's size (B_i) with the size of previous stratum $i+1$ (B_{i+1}) in an aggregate pyramid with d dimensions;

$$B_i = \frac{B_{i+1}}{2^d}$$

The size of an aggregate pyramid with two dimensions and the base stratum of size B ($B = b^d$ and $b = 2^n$ for d dimension and depth n) defined in Section 3.4.2.2 is

$$S = 1\frac{1}{3}B$$

By generalizing the formula for a d-dimensional aggregate pyramid ($d \geq 1$) with depth n and the size B (b^d where b is the number of bins along each dimension and $b = 2^n$) for the base stratum, we have

$$S = B + \frac{1}{2^d}B + \frac{1}{2^{2d}}B + \frac{1}{2^{3d}}B + \cdots + \frac{1}{2^{nd}}B$$

$$S = B\sum_{i=0}^{n}\frac{1}{2^{di}}$$

$$S \doteq 1\frac{1}{2^d - 1}B$$

The aggregate data pyramid is well behaved when the number of dimensions increase. In higher dimensions, the size of the aggregate pyramids grows gently compared to lower dimensions. Since a cubed aggregate pyramid is a cross product of single aggregate pyramids (axes), the size of a cubed pyramid with a number of axes $(axis_0 \dots axis_{a-1})$, axes' dimensions $d_0 \dots d_{a-1}$, and depth n when B is the size of the base of the Cubed aggregate pyramid can be calculated by following formula;

$$S = B \prod_{i=0}^{a-1} 1\frac{1}{2^{d_i} - 1} \tag{5.1}$$

where

$$\sum_{i=0}^{a-1} d_i = d$$

and

$$1\frac{1}{2^d - 1}B \leq B \prod_{i=0}^{a-1} 1\frac{1}{2^{d_i} - 1} \leq 2^d B$$

In the worse case, in constructing the cubed aggregate pyramid, each dimension is aggregated independently grouped by itself; it means that for total number of d dimensions there are d single axes. In this case, the size of the cubed pyramid is $2^d B$. In the best case, we have just one axis of d dimensions (a single aggregate pyramid) where its size is $1\frac{1}{2^d-1}B$. Assume we build a cubed aggregate pyramid of a 1D axis *time* and a 2D axis (x, y) over a given dataset with size B. The size of the 1D axis (one-dimensional aggregate pyramid) is 2B and the size of the 2D axis (two-dimensional aggregate pyramid) is $1\frac{1}{3}B$, the size of the computed cubed pyramid is then $2\frac{2}{3}B$ ($2 \times 1\frac{1}{3} \times B$).

5.5 Further Considerations

Depth of the aggregate pyramid is an important factor in building the pyramid efficiently. As mentioned in Chpater 3, Section 3.4.3.2.3, in the process of finding the natural depth, the calculated natural depth could be very deep that causes many bins to be islands (that is, having no immediate neighbor) in deep strata of the aggregate pyramid. In building cubed aggregate pyramids, the depth does also play an essential role to make this process efficient. It is possible to find the natural depth of each axes independently and to construct the cubed pyramid from axes at different depths. However, the existence of data points with non-empty neighbors in the base stratum of one axis does not guarantee same situation for other axes. The depth of one axis could be very deep for other axes and creates very sparse data on the base of the cubed aggregate pyramid. Hence, many data points becomes islands with no nearby non-empty bins, and they get replicated in the construction of cubed aggregate pyramid. And since, cubed pyramids are cross products of axes with different number of dimensions, the size of cubed pyramids grows much faster than aggregate pyramids in the presence of islands. Accelerating the growth of cubed pyramids' size imposes a negative impact on the performance in the construction process because the replicated data points are unnecessarily processed and repeated at higher strata.

In order to avoid the replication of islands and more efficiently build the cubed pyramids, we find the natural depth by considering all dimensions as a group, in the same way we process dimensions to find the depth for a single aggregate pyramid (axis). By processing all possible dimensions together to find the natural depth, the problem of islands' existence could be still a challenge depends on the calculated depth. Finding the natural depth for all dimensions together, however, moderates the acceleration of the size growth of the final product. After finding natural depth, we perform inductive aggregation along each group of dimensions which could be either a one-dimensional or muti-dimensional axes. Cubed aggregate pyramids can be studied as a single aggregate pyramid to define a proper depth. The probable number of islands in the computed natural depth from base-aggregation phase can be evaluated. If the likelihood of a large number of data points being islands is high in this depth, we can pick a higher stratum where there is a low probability of islands presence. By choosing an appropriate depth, cubed aggregate pyramids are well behaved and their size can be defined by Equation 5.1 in Section 5.4.

Chapter 6

Experiments

We applied inductive aggregation on the Brightkite Check-ins dataset [85] to construct a two-dimensional aggregate pyramid. In the base-aggregation phase, we process the raw data to build the base of the pyramid. Since we build the two-dimensional aggregate pyramid ($Pyramid(x, y)$), we aggregate data points to unify the duplicate tuples with same x and y coordinates. Therefore, the base tuples could contain aggregates of these duplicates. After we establish the base dataset from the raw data in the original dataset by pre-processing and pre-aggregating tuples (data points), a distinct Z-order value is assigned to each tuple at the base. The base dataset is then sorted based on the Z-order values. Next, we process the base to find the natural depth. Since the base dataset is sorted based on the Z-order values, each time we process two contiguous points on the disk including the current point and its preceding point. We apply the Super Bin Adjacency technique (Chapter 3, Section 3.4.3.2.2) to find the natural depth over the base dataset. Comparing the two-bit pairs of current point's Z-order value to the Z-order values of its

Stratum	**Points per Stratum**
1	4
2	15
3	40
4	101
5	262
6	649
7	1474
8	3085
9	6602
10	13962
11	27531
12	49545
13	83464
14	136000
15	210052
16	296490
17	388841
18	478680
19	555734
20	612096
21	646601
22	666077
23	678241
24	686178
25	688738
26	690303
27	691619
28	692736
29	693362

Table 6.1: Number of Data Points per Stratum of $2D$ Aggregate Pyramid (Brightkite dataset)

preceding point, we choose the greatest depth such that the current point is separated from its neighbor; call this the local natural depth of that point. The local natural depth of a point identifies the deepest stratum at which the point is separated from other points while it has at least one immediate none-empty neighbor. Moving up from a point's local natural depth to one stratum higher, the point is aggregated with one or more neighbors. The next local natural depth found in the iteration is then compared to the previous one to choose the deepest stratum. At the end of the process, the deepest stratum is found that guarantees the separation of all data points. In other words, the natural depth is the maximum calculated local natural depth determined for each data point.

To construct the two-dimensional aggregate pyramid of check-in data, we processed the raw data and found the depth 29 as the natural depth of the aggregate pyramid in the base-aggregation phase. In this phase, we applied aggregate functions to build the base of the pyramid. We then employed inductive aggregation to build the higher strata of the pyramid up to the apex, the highest stratum. The second column in Table 6.1 indicates the number of data points for each stratum of the aggregate pyramid. To build the first stratum after the base (stratum 28), the inductive aggregation is performed on the tuples at stratum 29. As a result of inductive aggregation, the number of tuples (bins) in stratum 28 has been reduced. The process of constructing the aggregate pyramid continues by applying the inductive aggregation over intermediate strata to build the very next higher stratum until the last stratum (with the lowest data resolution) is built. It can be observed from Table 6.1 that moving from stratum 29 up to stratum 1, the sizes of the strata reduces.

Figure 6.1 shows the data reduction from stratum 29 to the top of the pyramid. The number of data points slightly reduces from the base stratum up to stratum 22. After stratum 22, the data reduction of the intermediate strata accelerates between strata 22

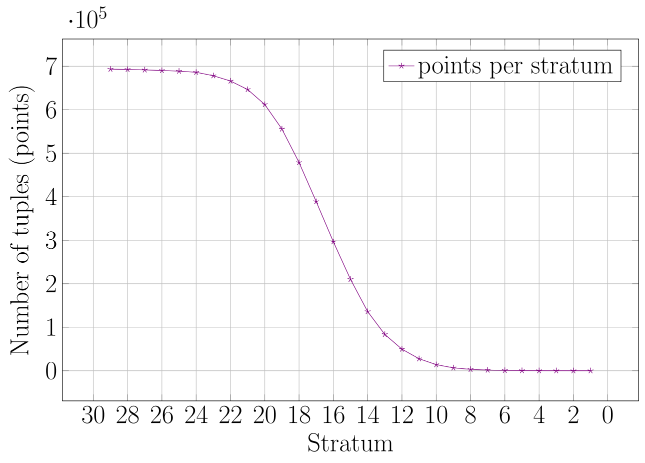


Figure 6.1: Strata Size Reduction of $2D$ Aggregate Pyramid (Brightkite dataset)

and 10. The slow data reduction in the depth of the pyramid implies that in very low strata there are a few points with immediate non-empty neighbors to be aggregated to form the very next higher stratum. To investigate the reason for slow changes in the size of lower strata in the aggregated pyramid, we examined the base dataset to determine the number of data points that get aggregated moving up from each stratum. To define that, we first need to know how many points there are with the local natural depth of each stratum. In the process of finding the natural depth, in each iteration, we find the local natural depth of every single data point. Using this information, we can define how many data points exist in every strata that are separated from other points (do not fall in the same bins) while they have at least one immediate neighbor.

Table 6.2 represents the information extracted from the base dataset collected in the process of finding the natural depth. These calculated numbers give us very precise

Stratum	**Points per Stratum**	**Distinct Points**	**Aggregate Points**
1	8	6	693361
2	27	19	693355
3	71	44	693336
4	182	111	693292
5	468	286	693181
6	1135	667	692895
7	2502	1367	692228
8	5178	2675	690861
9	11000	5822	688185
10	22918	11918	682363
11	43274	20355	670445
12	74430	31156	650089
13	121032	46602	618933
14	190430	69398	572331
15	279792	89361	502933
16	375042	95250	413571
17	469117	94075	318321
18	552519	83402	224246
19	615513	62994	140844
20	653958	38445	77850
21	673343	19385	39405
22	682707	9364	20020
23	687858	5151	10656
24	690832	2974	5505
25	691717	884	2531
26	692148	431	1646
27	692602	454	1215
28	693154	553	761
29	693363	209	209

Table 6.2: Aggregate Pyramid Statistics of Brightkite Dataset

information about the aggregate pyramid before building it. Column 4 of Table 6.2 represents the number of data points that are aggregated with other points moving one stratum up from each stratum. For instance, if we build the stratum 28 from stratum 29, it contains at most 209 aggregated data points and the rest are repeated data values. We then determined the number of distinct points in each stratum.

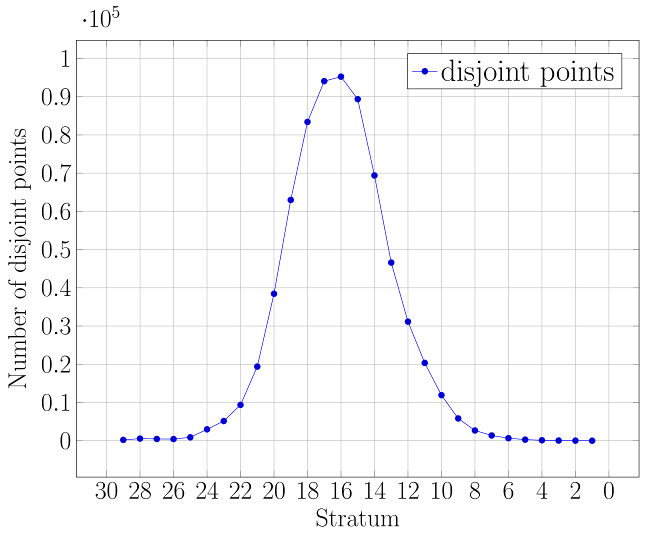


Figure 6.2: Calculated Number of Separate Points per Stratum

Figure 6.2 shows the distribution of disjoint data points in each stratum started from stratum 29. The result in cloumn 3, Table 6.2 states the number of points that are separate only in the correlated stratum and if you move one stratum up, those points are aggregated with their immediate neighbors from current stratum which is the definition of local natural depth. Depth 29, for instance, is the natural depth for 209 data points while the rest of the tuples (bins) are still disconnected from others at this stratum. When we move up to stratum 28, all 209 points from depth 29 get aggregated. These statistics have been collected in the process of finding local natural depth for each data point. Although we consider a natural depth deep enough to separate all data points in which no two points fall into the same bin, this depth could be too deep, causing many points to be far apart from each other with no non-empty nearby bins. As we see in

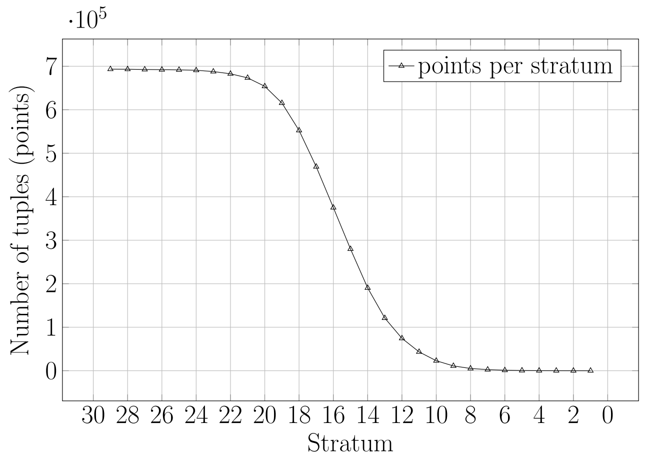


Figure 6.3: Calculated Number of Data Points per Stratum

Figure 6.2, there are a few points that get merged in the process of building the stratum 28.

Represented in Table 6.2, there are only 209 points in the stratum 29 that are aggregated in the next higher stratum and the rest, 693154 points, remains separated and are replicated in the stratum 28. In fact, when we apply inductive aggregation to summarize the stratum 29 (the base of the pyramid) to construct the stratum 28, not many non-empty bins are adjacent to be merged and aggregated at this depth. Thus, moving from stratum 29 up to stratum 28, we do not have considerable data reduction. Figure 6.3 shows very slow reduction from stratum 29 until depth 16. Moving from stratum 29 up to stratum 16, the number of individual points with non-empty neighbors increases. Therefore, in the higher strata of the aggregate pyramid, moving away from stratum 29, the number of bins with aggregate values escalates.

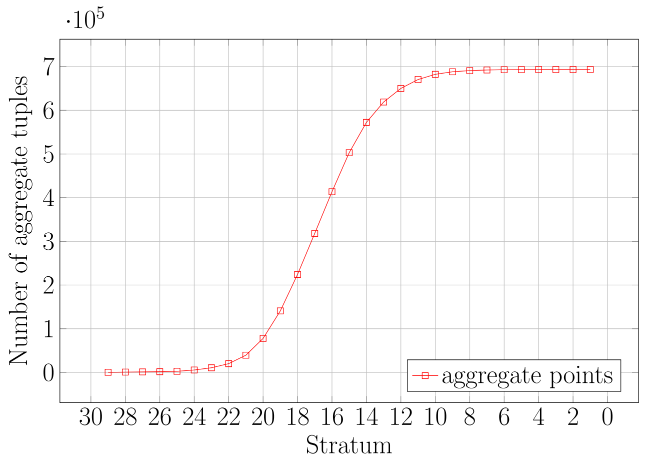


Figure 6.4: Calculated Number of Points with Aggregate Values per Stratum

Figure 6.4 represents the number of data points with aggregate values in each stratum. After depth 20, the points get aggregated with a higher rate. Reaching the high strata of the aggregate pyramid, the number of aggregated points reduces since there are smaller numbers of bins in very high strata on top of the pyramid. Figure 6.3 shows the size of each stratum with respect to the number of its tuples. Similarly, the same rate can be observed in the stratum size reduction. Up to stratum 20 started from depth 29, the size of the strata decreases slowly. After depth 20, the strata' size reduction accelerates and it slows down again after depth 10. We compared the final aggregate pyramid strata' size built from the actual check-in data with the statistics collected from the process of finding natural depth and obtained the similar result (Figure 6.5). In fact, in the process of finding the natural depth, we examine data points to find the local natural depth of each point, which is quite accurate, providing an acceptable approximation.

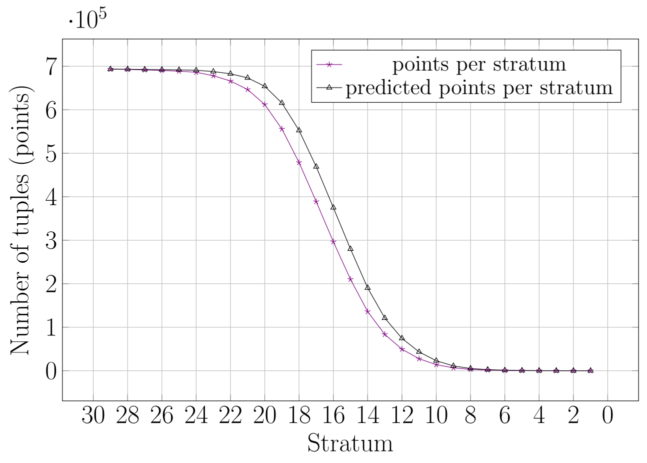


Figure 6.5: Strata Size Reduction of Implemented and (Predicted) Aggregate Pyramid

Since the natural depth could be deep enough to separate the points, it can result in having a large number of points with no immediate neighbors at the very low strata (higher resolution). Consequently, when we apply inductive aggregation on the tuples at the deep strata, many points do not get aggregated and they are replicated at higher strata causing a low rate of data reduction. We call the points with no immediate neighbors islands. Islands are bins such that their surrounding bins in the Z-order mapping curve are empty. An island can become adjacent to a non-empty bin in one stratum up or higher. By the time an island is transmitted to a higher stratum which attains at least one non-empty neighbor bin, it gets replicated in the process of inductive aggregation. Shown in Figure 6.3, we have many replicated points up to stratum 20 which determines that many bins are empty in the very deep strata resulting many points to become islands.

The repetition of the islands in deep strata of the aggregate pyramid influences performance. To improve the performance, the aggregate pyramid can be built from higher strata with less number of islands. To determine the proper depth with more satisfying reduction, we can define a threshold for the depth at which we have more data reduction compared to the calculated natural depth. To find the threshold depth, we can approximately model the size reduction with a polynomial. We can differentiate the approximate polynomial model to calculate the first and second derivatives of that function to find the points in which we have decent reduction. The first column in Table 6.2 can be approximately modeled with a polynomial function. If we model the size reduction graph with a third degree polynomial ($f(x)$), for instance, the point with $x = 16$ is the local maximum point of its first derivative. The graph representing the first derivative of f(x) is similar to the graph presented in Figure 6.2. Figure 6.2 shows the difference of each stratum with the next higher stratum. In this graph, we observe that in depth 16 there exist many points with non-empty neighbors causing the acceptable data reduction compared to the size of the base. To find the local maximum of the approximate polynomial function, the solution of the second derivative of the function f(x) can be examined to find the local maximum points which for a third degree polynomial is approximately stratum 16 ($x = 16$). The critical point of the function $f'(x)$ can be considered as the threshold depth at which we observe adequate data reduction. In the case, that there exist more than one local maximum, we select the highest one as the threshold depth.

Likewise, we employed the same technique on Brightkite dataset to build a $1D \times 2D$ aggregate pyramid; $Pyramid(time) \times Pyramid(x, y)$. The result of the experiment

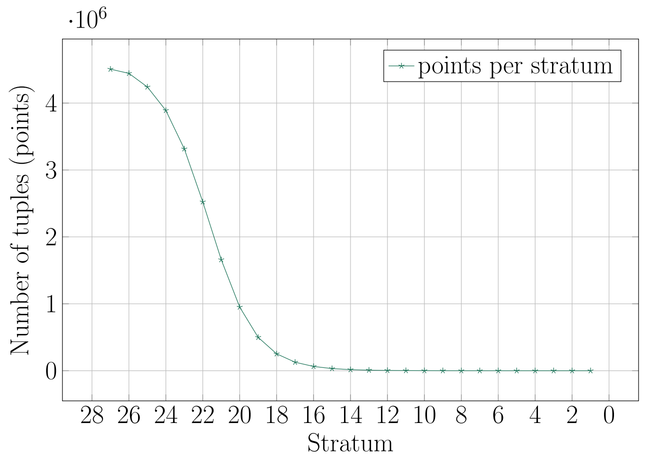
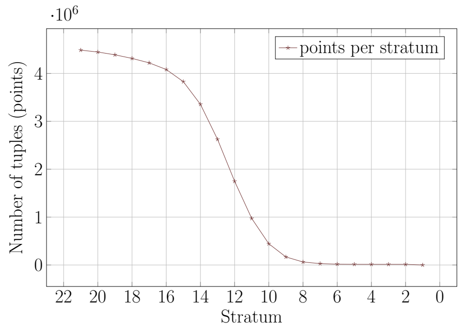
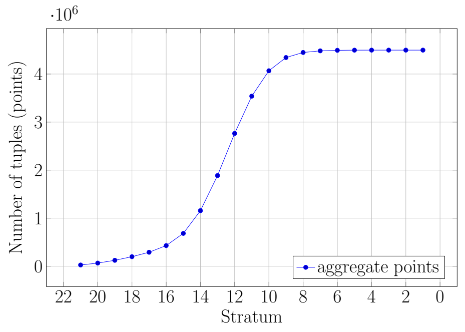


Figure 6.6: Strata Size Reduction of $1D$ Aggregate Pyramid

is uniform with the result achieved from the previous observation of building the $2D$ aggregate pyramid. Considering three dimensions x, y, and $time$, the natural depth for the $1D \times 2D$ pyramid was determined as depth 21. The computed natural depth for $2D$ pyramid $Pyramid(x, y)$ and $1D$ pyramid $Pyramid(time)$ are respectively 29 and 27. It is obvious from Figure 6.6 that stratum size is decreased from depth 27 with a satisfying reduction rate in $1D$ aggregate pyramid, $Pyramid(time)$. The statistics of $1D$ aggregate pyramid shows that most of the points have at least one immediate non-empty bin at depth of the pyramid, and, as a result, we do not have many islands in very low strata starting from the threshold natural depth, depth 27. In the $1D \times 2D$ aggregate pyramid ($Pyramid(time) \times Pyramid(x, y)$), however, we observe very slow reduction between strata 21 and 16 (see Figure 6.7a). The determined natural depth for $1D \times 2D$ aggregate pyramid is 21. With same reasoning, the presence of many islands in the depth



(a) Strata Size Reduction of $1D \times 2D$ Aggregate Pyramid1



(b) Data Points with Aggregate Values per Stratum

Figure 6.7: Statistics of $1D \times 2D$ Aggregate Pyramid of Brightkite Dataset

of the aggregate pyramid is caused the size of the lower strata to be slightly decreased. Presented in Figure 6.7b, the number of aggregate points at the depth of the aggregate pyramid is small. Therefore, we observe less reduction before depth 16, knowing there are many islands in deep strata of the aggregate pyramid. After depth 16, the observation shows that the number of aggregate points increases which causes the size of the strata to be reduced with a satisfying speed.

We also applied inductive aggregation on NYC Taxi Trips dataset [86] to construct the $1D \times 2D \times 2D$ aggregate pyramid, $Pyramid(time) \times Pyramid(x, y) \times Pyramid(u, v)$. Each data point has 5 dimensions; time, source location (x,y), and destination location (u,v). In the same manner, we determine the natural depth of the $1D \times 2D \times 2D$ aggregate pyramid by sampling points and estimating the statistics corresponding the number of separate points in each stratum during the process of finding the local natural depth of

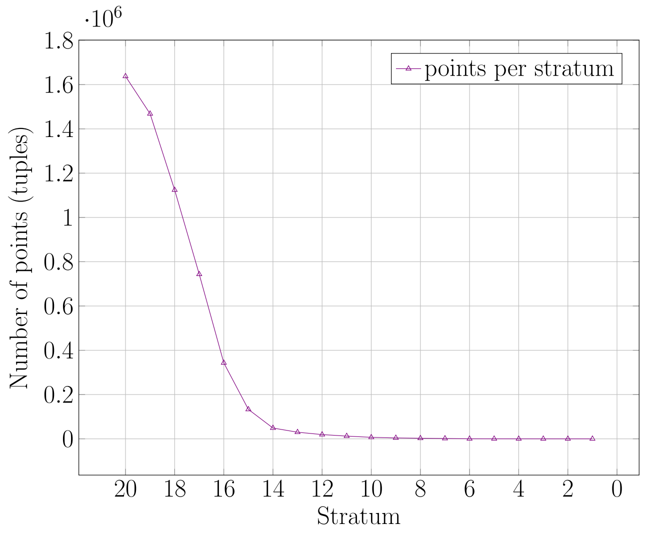


Figure 6.8: Strata Size Reduction of $1D \times 2D \times 2D$ Aggregate Pyramid

each point. To calculate the depth of the pyramid, we employ super bin adjacency on the original dataset. We considered the size of each dimension a 20-bit number. Therefore, the size of each dimension in the base stratum is 2^{20} and the stratum size is $2^{5\times20}$. Since the Z-order value of a point in a multi-dimensional space is the result of the bit interleaving of its dimensions, the Z-order of each tuples in the $1D \times 2D \times 2D$ aggregate pyramid is a 100-bit positive number from 0 to $2^{100} - 1$ (which is very large). The natural depth calculated for the $1D \times 2D \times 2D$ aggregate pyramid of NYC Taxi Trips data is 20. The depth 20 is the deepest stratum that could be calculated since our dimensions are 20-bit numbers. If we build the $1D \times 2D \times 2D$ aggregate pyramid before this depth, we observe that the number of islands below this depth is increased. Figure 6.8 expresses the satisfying reduction from depth 20. The result emphasizes that most of the points have at least one non-empty neighbor at depth 20. Building the pyramid from any stratum below stratum 20 would cause a repetition of islands; any deeper the base of the pyramid is built, more disconnect points are replicated.

Chapter 7

Conclusions and Future Work

7.1 In Summary

The objective of this thesis was to provide a hierarchical structure defined in a database and managed by a database system in the back-end to maintain a tight coupling between visualization systems as the front-end. A tightly coupled system can benefit from the effective and powerful features and techniques of the database system to facilitate and improve the interactive operations in data visualization. In this thesis, we have contributed following methods and techniques to create a hierarchical structure to support interactive data visualization of large datasets.

The concept of data compression and multi-scale representation of data has been adopted in various domains such as image processing, computer graphics, and computer vision. The original idea was taken from image processing techniques such as progressive transmission and multi-scale representation of images. In image processing, for instance, JPEG2000 maintains progressive and hierarchical modes. In the progressive mode, image data is gradually visualized by starting from very low quality version which is an approximation of the initial image, then refining the image as more data is received until the desired resolution is obtained. In the hierarchical mode, on the other hand, different scales of an image are created at different sizes and resolutions. The version compatible with the screen size and resolution is then sent to be visualized. This technique is known as pyramidial image representation or pyramidial mode. In this mode, the initial image is down-sampled into a lower quality version with half of the resolution of the original image. The lower resolution sample is filtered and sub-sampled again to form the next lower resolution copy, and so forth. This mode also supports image compression and progressive transmission.

Likewise, in visualizing large datasets, data compression and reduction techniques can be employed to deal with issues such as over plotting when the number of data points is larger than the number of pixels on the screen. Similar to compression techniques in image processing, reduction techniques can be applied on the initial dataset to summarize the data and to scale data to the desired resolution. However, for each request, the initial dataset has to be examined to provide the sufficient response. Besides data reduction techniques, other methods can be adapted to improve the performance in interactive data visualization. To reduce the processing time, data can be processed in advance and multiple resolutions of the dataset can be prepared. Data reduction techniques can be iteratively applied on the original dataset or reduced versions to achieve a compact and successive structure representing multiple levels of detail.

Hierarchical structures such as quadtrees have garnered tremendous attention in spatial databases. The term quadtree has been adopted to describe a class of hierarchical structures which are based on the recursive-decomposition principle [25]. Another structure related to quadtrees is the pyramids. The pyramid provides a hierarchical data representation where different resolutions of an image are successively arranged from the highest quality version to the lowest quality copy. In an image pyramid, for instance, the lower quality version of an image can be examined to search for a feature instead of processing the initial large image [23]. Hierarchical structures are prominent data structures in spatial databases to index multi-dimensional datasets [95].

To process and search high-dimensional datasets, data can be mapped to one dimensional data points. One-dimensional datasets impose less complexity by simplifying some processes such as range queries. By mapping a multi-dimensional dataset to a one-dimensional domain using space filling curves [29], a unique number is assigned to each data point in a sequence that preserve the locality of points in the space. The data can then be sorted and stored by the assigned values in a linear order. An efficient one-dimensional *B+-tree* can be used to improve the performance. *B+-tree* indexes efficiently support range queries. Since the tuples (data points) are sequentially sorted where two spatially adjacent points are also likely to be close on the disk, a limited scope of data can be searched and queried by scanning continuous tuples.

To maintain the tight coupling between databases and visualization systems, we proposed a hierarchical structure and methods to build and query efficiently the proposed structure. Adopted techniques in this thesis empower the integration of database systems and visualization techniques where the powerful features of the database system for data manipulation and storage can be used to improve the visualization process by pushing data management into the database engine. Our proposed hierarchical structure, the *inductive-aggregate pyramid* provides a multi-level representation of a given data set. The multi-dimensional ordering (Z-order) employed in our structure facilitates the processing and indexing of data to materialize and query the aggregate pyramid. The aggregate pyramid efficiently supports interactive operations. The generalized form of the aggregate pyramid, *the cubed pyramid*, maintains richer representation of high-dimensional datasets. A summary of our contributions follows.

7.2 Review of Contributions

The main contributions of the thesis is stated as follows.

1. Provided a comprehensive survey of state of the art of database support for interactive visualization.

2. Proposed a hierarchical data structure called the *inductive aggregate pyramid* that provides multi-scale representation of data to support interactive exploration and visualization of large datasets. The reduction techniques along with multi-dimensional ordering methods adopted in a recursive procedure summarizes data in a successive and compact form.

(a) Proposed techniques and principles that iteratively aggregate data to construct multiple representations of a dataset, called *inductive aggregation*. In inductive aggregation, aggregate functions are iteratively and efficiently performed on data quadrants to form the lower levels of detail by sequentially scanning the last computed level. Inductive aggregation spatially aggregates data over one dimension or a group of dimensions.

(b) Defined the meaningful depth for the aggregate pyramid. We define how deep an aggregate pyramid should be constructed to provide useful and meaningful detail. The depth of the pyramid is an essential element in building the aggregate pyramid. Since the aggregate pyramid provides multiple levels of resolution, it needs to maintain a high level of detail representing single data points to very low resolution data (more aggregate values) that can be extracted and visualized. The natural depth is defined as a depth at which all data points are separated; below this depth there is no more detail to be extracted.

(c) Designed API to support interactive visual operations. We propose algorithms and techniques that interactively query the aggregate pyramid. In these proposed techniques, the quadrant recursive property of Z ordering [20] (the space filling curve used in our structure) is employed to query efficiently the aggregate pyramid.

3. Implemented efficiently the aggregate pyramid materialization and use of inductive

aggregation.

(a) Materializing efficiently the aggregate pyramid. The construction of the aggregate pyramid is performed in two phases: *base-aggregation* and *re-aggregation*. In the base, the initial dataset is processed and the sequential Z-order values are assigned to tuples. In the next phase, inductive aggregation efficiently aggregates groups of four consecutive tuples by performing a sequential scan on the disk.

(b) Indexed the aggregate pyramid for fast API evaluation. The aggregate pyramid is indexed using the *B+-tree* index structure on the level of detail (stratum) and the multi-dimensional ordering key values (Z-order [20]) to facilitate the constructing and querying this structure.

(c) Determined efficiently the natural depth and demonstrating its critical impact on the efficiency of building the aggregate pyramid. We proposed an algorithm that calculates the *natural depth* of the aggregate pyramid.

(d) Evaluated API cost at interactive speed. The aggregate pyramid is queried efficiently to support interactive operations. With the proposed techniques, to answer a query, at most four probes are needed to be examined and the size of final result is at most four times bigger than the initial query window.

4. Generalized aggregate pyramids to cubed pyramids. A novel structure, the *cubed pyramid*, was proposed which is a generalization of roll-up data cubes. The cubed

pyramid is a cross product of aggregate pyramids. This structure provides hierarchical building and processing of data over one or more desired dimensions, in contrast to data cubes which aggregates data over all combinations of dimensions.

   (a) Defined multiple axes for richer representation of high-dimensional data. An aggregate pyramid can be considered as an axis of one or more dimensions. A cubed pyramid consists of one or more axes.
   (b) Implemented efficiently materialization and use of cubed pyramids. The depth of each axis is computed by the same technique used in the aggregate pyramid construction. Inductive aggregation is then applied over each axis while the attributes of other axes are held constant.
   (c) Demonstrated how cubed pyramids' richer presentation supports richer interactive data representations. Aggregating data over pre-defined dimensions (axes) provides an independent axis exploration. Different resolutions of one or more axes can be displayed while the resolution of other axes is fixed.

5. Verified experimentally efficiency of aggregate pyramids

   (a) Estimated the aggregate pyramid cardinality in advance. Through the process of finding the natural depth, the size of the final cardinality can be predicted. Since in this technique, the natural depth is calculated by finding the local natural depth of sampled data points, the size of each stratum can also be approximately computed. These statistics provide valuable information to study and analyze the performance of the aggregate pyramid construction.

(b) Showed the effect of natural depth on cost. The experiments show that in a very deep stratum where data is sparse, performing aggregation to the next higher stratum does not result in the desired data reduction. Building an aggregate pyramid very deep can result in many "islands." Islands are bins with no immediate non-empty neighbors. They are replicated without being aggregated with other points in the inductive aggregation process at higher strata.

7.3 Future Work

- Evaluate the performance of interactive operations over cubed pyramids, especially when the cubed pyramid is made of more than two axes.
- Resolve the inefficiencies caused by islands. Islands impose a negative impact on the efficiency of constructing the aggregate pyramid, and especially more so on the cubed pyramids. Finding a solution to this problem will improve the efficiency of aggregate and cubed pyramids construction.
    - Currently, we aggregate data over non-overlapping fixed-size aggregate windows (data quadrants), which are composed of 2×2 squares containing four bins (tuples) for a two-dimensional dataset, in inductive aggregation. Aggregating data over variable-size aggregate windows within a stratum or between

related strata can be investigated to deal with the problem of islands' replication. Since islands do not have immediate neighbors to aggregate with, a larger size aggregate window could include those islands preventing them to replicate.

- In this thesis, we have adopted the pyramidial structure. As mentioned before in Chapter 3, pyramids and quadtrees are relevant structures, while having some differences. In a quadtree, terminal nodes can have different levels (depths) in contrast to a pyramid where all terminal nodes, representing the most detailed values, have the deepest level. Pyramids can be defined as complete region quadtrees [26, 96]. A region quadtree [97] is a $2^n \times 2^n$ area which is successively subdivided into four equal quadrants. The region is subdivided into quadrants, sub-quadrants, sub-sub-quadrants, and so forth until indivisible units are obtained (which represents single data points or pixels). Since islands do not aggregate in very deep strata, their local natural depth is different from the natural depth calculated for the aggregate pyramid. Instead of reproducing the islands in deep strata where they do not have any non-empty nearby bins, we could represent them by their own local natural depth which could be higher than the depth of the aggregate pyramid. Unlike our current structure where all terminal nodes have the same depth which is also the depth of the whole structure, we might have various depths in the hierarchical structure.

  - The idea of not replicating islands would sufficiently improve the performance of constructing the aggregate and cubed pyramids. However, the lack of presence of islands in the deeper strata imposes complication on the interactive operations. In our current approach, because all tuples in a stratum are present and sorted on the Z-order values, a desired resolution can be efficiently queried over a single stratum traversing a range of consecutive numbers. In querying the deep strata where some points might not be presented to prevent the replication of islands, more strata may be required to be examined. One or more strata above the examined stratum have to be included to provide the complete result in response to a request. Therefore, the extra work needed to search and fetch the related tuples from corresponding strata can influence the performance of interactive operations.

- Apply other data reduction techniques to summarize data. In this study, we have applied aggregation as our data reduction technique to generate the reduced version of a dataset at different levels of detail. We are interested in studying the possibility of employing different reduction techniques such as sampling and determining how the aggregate and cubed pyramids can be interpreted under these reduction techniques.

- Interpolate over bins (thus approximating) to improve performance. In this study, we assume that the window of interest as a rectangular window ($n \times m$) that includes full size bins. An approximate or exact answer can then be prepared by employing the quadrant-recursive property of the Z-order values since we map the current window to a higher strata to query at most by four probes. However, a window of interest may not completely cover all bins. Indeed, a circular or rectangular window of interest might partially cover some bins on the border. One could investigate how a partial value could be calculated to represent the fractional area based on the portion of a bin that is covered by the window of interest. It has to be also determined what aggregate values (min, max, mean, etc.) can be sufficiently interpolated in this process.

7.4 In Closing

Visual exploration and representation of data is an effective tool employed to facilitate the process of knowledge discovery and decision making. As technologies evolve, more powerful and effective techniques are being developed. However, the fast growth of data and the need to store and process more information have imposed more challenges and complications for data visualization and exploration processes. Integration of database systems with visualization tools provide very effective techniques to overcome these problems since database systems are well designed to support data manipulation and storage of very large datasets. The database community has done a remarkable job of incorporating many novel ideas and effective techniques to empower interactive visual data exploration and processing. However, there are many issues and difficulties that have not been addressed sufficiently. To enhance the process of visual representation of large datasets, there are many potential approaches that can be developed and employed to facilitate this process, especially following the integration of database systems and data visualization techniques.

Bibliography

[1] Stuart K Card and Jock Mackinlay. The structure of the information visualization design space. In *Information Visualization, 1997. Proceedings., IEEE Symposium on*, pages 92–99. IEEE, 1997.

[2] Stephen Few. Data visualization for human perception. *The Encyclopedia of Human-Computer Interaction, 2nd Ed.*, 2013.

[3] Alan M MacEachren and John H Ganter. A pattern identification approach to cartographic visualization. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 27(2):64–81, 1990.

[4] Dulclerci Sternadt Alexandre and Joao Manuel RS Tavares. Introduction of human perception in visualization. 2010.

[5] Shusen Liu, Dan Maljovec, Bei Wang, Peer-Timo Bremer, and Valerio Pascucci. Visualizing high-dimensional data: Advances in the past decade. In *Proc. Eurographics Conf. Visualization*, pages 20151115–127, 2015.

[6] Georges Grinstein, Marjan Trutschl, and Urska Cvek. High-dimensional visualizations. In *Proceedings of the Visual Data Mining Workshop, KDD*. Citeseer, 2001.

[7] Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data mining and knowledge discovery*, 1(1):29–53, 1997.

[8] Tableau software. `http://www.tableausoftware.com`.

[9] Richard Wesley, Matthew Eldridge, and Pawel T Terlecki. An analytic data engine for visualization in tableau. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 1185–1194. ACM, 2011.

[10] Richard Michael Grantham Wesley and Pawel Terlecki. Leveraging compression in the tableau data engine. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 563–573. ACM, 2014.

[11] Parke Godfrey, Jarek Gryz, and Piotr Lasek. Interactive visualization of large data sets. 2015.

[12] Leilani Battle, Michael Stonebraker, and Ronald Chang. Dynamic reduction of query result sets for interactive visualizaton. In *Big Data, 2013 IEEE International Conference on*, pages 1–8. IEEE, 2013.

[13] Niklas Elmqvist and Jean-Daniel Fekete. Hierarchical aggregation for information visualization: Overview, techniques, and design guidelines. *Visualization and Computer Graphics, IEEE Transactions on*, 16(3):439–454, 2010.

[14] Chris Stolte, Diane Tang, and Pat Hanrahan. Multiscale visualization using data cubes. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):176–187, 2003.

[15] Zhicheng Liu, Biye Jiang, and Jeffrey Heer. immens: Real-time visual querying of big data. In *Computer Graphics Forum*, volume 32, pages 421–430. Wiley Online Library, 2013.

[16] Hadley Wickham. Bin-summarise-smooth: a framework for visualising large data, 2013.

[17] Lauro Lins, James T Klosowski, and Carlos Scheidegger. Nanocubes for real-time exploration of spatiotemporal datasets. *Visualization and Computer Graphics, IEEE Transactions on*, 19(12):2456–2465, 2013.

[18] Ben 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.

[19] Alexandre Perrot, Romain Bourqui, Nicolas Hanusse, Frédéric Lalanne, and David Auber. Large interactive visualization of density functions on big data infrastructure. In *Large Data Analysis and Visualization (LDAV), 2015 IEEE 5th Symposium on*, pages 99–106. IEEE, 2015.

[20] Guy M Morton. *A computer oriented geodetic data base and a new technique in file sequencing.* International Business Machines Company New York, 1966.

[21] Yvette E Gelogo, Tai-hoon Kim, and Bimal Kumar Ray. Compressed images transmission issues and solutions. 2014.

[22] David Taubman and Michael Marcellin. *JPEG2000 Image Compression Fundamentals, Standards and Practice: Image Compression Fundamentals, Standards and Practice*, volume 642. Springer Science & Business Media, 2012.

[23] Steven Tanimoto and Theo Pavlidis. A hierarchical data structure for picture processing. *Computer Graphics and Image Processing*, 4(2):104–119, 1975.

[24] Stefan Berchtold, Christian Böhm, and Hans-Peter Kriegal. The pyramid-technique: towards breaking the curse of dimensionality. In *ACM SIGMOD Record*, volume 27, pages 142–153. ACM, 1998.

[25] Hanan Samet. An overview of quadtrees, octrees, and related hierarchical data structures. In *Theoretical Foundations of Computer Graphics and CAD*, pages 51–68. Springer, 1988.

[26] Hanan Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys (CSUR)*, 16(2):187–260, 1984.

[27] Irene Gargantini. An effective way to represent quadtrees. *Communications of the ACM*, 25(12):905–910, 1982.

[28] Clifford A Shaffer and Hanan Samet. Algorithm to expand regions represented by linear quadtrees. *Image and Vision Computing*, 6(3):162–168, 1988.

[29] Giuseppe Peano. Sur une courbe, qui remplit toute une aire plane. *Mathematische Annalen*, 36(1):157–160, 1890.

[30] David M Mark. Neighbor-based properties of some orderings of two-dimensional space. *Geographical Analysis*, 22(2):145–157, 1990.

[31] Jock Mackinlay. Automating the design of graphical presentations of relational information. *Acm Transactions On Graphics (Tog)*, 5(2):110–141, 1986.

[32] Steven F Roth and Joe Mattis. Automating the presentation of information. In *Artificial Intelligence Applications, 1991. Proceedings., Seventh IEEE Conference on*, volume 1, pages 90–97. IEEE, 1991.

[33] Christopher Ahlberg and Ben Shneiderman. Visual information seeking: Tight coupling of dynamic query filters with starfield displays. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 313–317. ACM, 1994.

[34] Christopher Ahlberg and Erik Wistrand. Ivee: An information visualization and exploration environment. In *Information Visualization, 1995. Proceedings.*, pages 66–73. IEEE, 1995.

[35] NK Jogt and B Shneiderman. Starfield visualization with interactive smooth zooming. *Visual Database Systems 3: Visual information management*, page 1, 1995.

[36] Christopher Ahlberg, Christopher Williamson, and Ben Shneiderman. Dynamic queries for information exploration: An implementation and evaluation. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 619–626. ACM, 1992.

[37] Christopher Ahlberg. Spotfire: an information exploration environment. *ACM SIGMOD Record*, 25(4):25–29, 1996.

[38] Jade Goldstein, Steven F Roth, John Kolojejchick, and Joe Mattis. A framework for knowledge-based interactive data exploration. *Journal of Visual Languages & Computing*, 5(4):339–363, 1994.

[39] Alexander Aiken, Jolly Chen, Michael Stonebraker, and Allison Woodruff. Tioga-2: A direct manipulation database visualization environment. In *icde*, pages 208–217, 1996.

[40] Michael Stonebraker, Jolly Chen, Nobuko Nathan, Caroline Paxson, and Jiang Wu. Tioga: Providing data management support for scientific visualization applications. In *VLDB*, volume 93, pages 25–38, 1993.

[41] Amihai Motro. Flex: A tolerant and cooperative user interface to databases. *Knowledge and Data Engineering, IEEE Transactions on*, 2(2):231–246, 1990.

[42] Daniel A Keim and Hans-Peter Kriegel. Visdb: Database exploration using multidimensional visualization. *Computer Graphics and Applications, IEEE*, 14(5):40–49, 1994.

[43] Steven F Roth, Peter Lucas, Jeffrey A Senn, Cristina C Gomberg, Michael B Burks, Philip J Stroffolino, AJ Kolojechick, and Carolyn Dunmire. Visage: a user interface environment for exploring information. In *Information Visualization'96, Proceedings IEEE Symposium on*, pages 3–12. IEEE, 1996.

[44] Jade Goldstein and Steven F Roth. Using aggregation and dynamic queries for exploring large data sets. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 23–29. ACM, 1994.

[45] Mark Derthick, John Kolojejchick, and Steven F Roth. An interactive visual query environment for exploring data. In *Proceedings of the 10th annual ACM symposium on User interface software and technology*, pages 189–198. ACM, 1997.

[46] Miron Livny, Raghu Ramakrishnan, Kevin Beyer, Guangshun Chen, Donko Donjerkovic, Shilpa Lawande, Jussi Myllymaki, and Kent Wenger. Devise: integrated querying and visual exploration of large datasets. In *ACM SIGMOD Record*, volume 26, pages 301–312. ACM, 1997.

[47] Robert Bosch, Chris Stolte, Diane Tang, John Gerth, Mendel Rosenblum, and Pat Hanrahan. Rivet: A flexible environment for computer systems visualization. *ACM SIGGRAPH Computer Graphics*, 34(1):68–73, 2000.

[48] Anna Fredrikson, Chris North, Catherine Plaisant, and Ben Shneiderman. Temporal, geographical and categorical aggregations viewed through coordinated displays: a

case study with highway incident data. In *Proceedings of the 1999 workshop on new paradigms in information visualization and manipulation in conjunction with the eighth ACM internation conference on Information and knowledge management*, pages 26–34. ACM, 1999.

[49] Joseph M Hellerstein, Peter J Haas, and Helen J Wang. Online aggregation. In *ACM SIGMOD Record*, volume 26, pages 171–182. ACM, 1997.

[50] Uwe Jugel, Zbigniew Jerzak, Gregor Hackenbroich, Gregor Hackenbroich, and Volker Markl. M4: a visualization-oriented time series data aggregation. *Proceedings of the VLDB Endowment*, 7(10):797–808, 2014.

[51] Jean-François Im, Felix Giguere Villegas, and Michael J McGuffin. Visreduce: Fast and responsive incremental information visualization of large datasets. In *Big Data, 2013 IEEE International Conference on*, pages 25–32. IEEE, 2013.

[52] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[53] Ben Shneiderman. Extreme visualization: squeezing a billion records into a million pixels. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 3–12. ACM, 2008.

[54] Brian Johnson and Ben Shneiderman. Tree-maps: A space-filling approach to the visualization of hierarchical information structures. In *Visualization, 1991. Visualization'91, Proceedings., IEEE Conference on*, pages 284–291. IEEE, 1991.

[55] Chris Stolte, Diane Tang, and Pat Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *Visualization and Computer Graphics, IEEE Transactions on*, 8(1):52–65, 2002.

[56] Bill Jelen and Michael Alexander. *Pivot Table Data Crunching: Microsoft Excel 2010*. Pearson Education, 2010.

[57] Chris Stolte, Diane Tang, and Pat Hanrahan. Query, analysis, and visualization of hierarchically structured data using polaris. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 112–122. ACM, 2002.

[58] Parke Godfrey, Jarek Gryz, Piotr Lasek, and Nasim Razavi. Visualization through inductive aggregation. In *Proceedings of EDBT*, 2016.

[59] Iosif Lazaridis and Sharad Mehrotra. Progressive approximate aggregate queries with a multi-resolution tree structure. In *ACM SIGMOD Record*, volume 30, pages 401–412. ACM, 2001.

[60] Hanan Samet. Spatial data structuresi. 1995.

[61] Walid G Aref and Hanan Samet. Efficient processing of window queries in the pyramid data structure. In *Proceedings of the ninth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 265–272. ACM, 1990.

[62] Hanan Samet. *Foundations of multidimensional and metric data structures*. Morgan Kaufmann, 2006.

[63] A Rosenfeld. Some useful properties of pyramids. In *Multiresolution Image Processing and Analysis*, pages 2–5. Springer, 1984.

[64] Hanan Samet. Hierarchical spatial data structures. In *Design and Implementation of Large Spatial Databases*, pages 191–212. Springer, 1989.

[65] DM Mark and MF Goodchild. On the ordering of two-dimensional space: introduction and relation to tesseral principles. *Spatial Data Processing using Tesseral Methods. NERC, Unit for Thematic Information Systems, Natural Environment Research Council, Swindon, Great Britain*, pages 179–192, 1986.

[66] Hans Sagan. *Space-filling curves*. Springer Science & Business Media, 2012.

[67] David Hilbert. Ueber die stetige abbildung einer line auf ein flächenstück. *Mathematische Annalen*, 38(3):459–460, 1891.

[68] David M Mark and Jean Paul Lauzon. Linear quadtrees for geographic information systems. In *Proceedings of the International Symposium on Spatial Data Handling*, volume 2, pages 412–430. Zurich, 1984.

[69] David M Mark. The use of quadtrees in geographic information systems and spatial data handling. In *Proceedings Auto Carto London*, volume 1, pages 517–526, 1986.

[70] Jack A Orenstein. Algorithms and data structures for the implementation of a relational database system. 1983.

[71] Jack A Orenstein and Tim H Merrett. A class of data structures for associative searching. In *Proceedings of the 3rd ACM SIGACT-SIGMOD symposium on Principles of database systems*, pages 181–190. ACM, 1984.

[72] Marvin White. N-trees: large ordered indexes for multi-dimensional space. *Application Mathematics Research Sta, Statistical Research Division, US Bureau of the Census*, 1981.

[73] Michael F Goodchild and Andrew W Grandfield. Optimizing raster storage: an examination of four alternatives. In *Proceedings of Auto-Carto*, volume 6, pages 400–407, 1983.

[74] Zuotao Li, X Sean Wang, Menas Kafatos, and Ruixin Yang. A pyramid data model for supporting content-based browsing and knowledge discovery. In *Scientific and Statistical Database Management, 1998. Proceedings. Tenth International Conference on*, pages 170–179. IEEE, 1998.

[75] Rashid Ansari and Nasir Memon. The jpeg standard. *Department of Computer Science, University of Illinois and Polytechnic University, Chicago & New York*, 1999.

[76] Gregory K Wallace. The jpeg still picture compression standard. *Consumer Electronics, IEEE Transactions on*, 38(1):xviii–xxxiv, 1992.

[77] Brani Vidakovic. Discrete wavelet transformation. *Statistical Modeling by Wavelets*, pages 101–117.

[78] Charilaos Christopoulos, Athanassios Skodras, and Touradj Ebrahimi. The jpeg2000 still image coding system: an overview. *Consumer Electronics, IEEE Transactions on*, 46(4):1103–1127, 2000.

[79] Alfred Haar. Zur theorie der orthogonalen funktionensysteme. *Mathematische Annalen*, 69(3):331–371, 1910.

[80] Eric J Stollnitz, Tony D DeRose, and David H Salesin. Wavelets for computer graphics: a primer. *Computer Graphics and Applications, IEEE*, 15(3):76–84, 1995.

[81] Prashan Premaratne. Effective hand gesture classification approaches. In *Human Computer Interaction Using Hand Gestures*, pages 105–143. Springer, 2014.

[82] Joseph M Hellerstein, Ron Avnur, Andy Chou, Christian Hidber, Chris Olston, Vijayshankar Raman, Tali Roth, and Peter J Haas. Interactive data analysis: The control project. *Computer*, 32(8):51–59, 1999.

[83] Jeffrey Heer, Jock D Mackinlay, Chris Stolte, and Maneesh Agrawala. Graphical histories for visualization: Supporting analysis, communication, and evaluation. *Visualization and Computer Graphics, IEEE Transactions on*, 14(6):1189–1196, 2008.

[84] Jock D Mackinlay, Pat Hanrahan, and Chris Stolte. Show me: Automatic presentation for visual analysis. *Visualization and Computer Graphics, IEEE Transactions on*, 13(6):1137–1144, 2007.

[85] Brightkite Dataset. `https://snap.stanford.edu/data/loc-brightkite.html`.

[86] TLC Trip Record Dataset. `http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml`.

[87] Carlos Scheidegger. Interactive visual analysis of big data. *Handbook of Big Data*, page 61, 2016.

[88] Yannis Sismanis, Antonios Deligiannakis, Nick Roussopoulos, and Yannis Kotidis. Dwarf: Shrinking the petacube. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 464–475. ACM, 2002.

[89] Y Bedard. Spatial olap. vidéo-conférence. 2ème forum annuel sur la rd, géomatique vi: Un monde accessible, montréal, 1997.

[90] Sonia Rivest, Yvan Bédard, and Pierre Marchand. Toward better support for spatial decision making: defining the characteristics of spatial on-line analytical processing (solap). *GEOMATICA-OTTAWA-*, 55(4):539–555, 2001.

[91] Rosa Matias and Joao Moura-Pires. Spatial on-line analytical processing (solap): A tool the to analyze the emission of pollutants in industrial installations. In *Artificial intelligence, 2005. epia 2005. portuguese conference on*, pages 214–217. IEEE, 2005.

[92] Nurefsan Gur, Katja Hose, Torben Bach Pedersen, and Esteban Zimanyi. Modeling and querying spatial data warehouses on the semantic web. *Lecture Notes in Computer Science*, 2015.

[93] Marcus Sampaio, C Baptista, A Souza, and F Nascimento. Enhancing decision support systems with spatial capabilities. *Intelligent Databases: Technologies and Applications. Hershey PA*, 17033:94–116, 2006.

[94] Pierre Marchand, Alexandre Brisebois, Yvan Bédard, and Geoffrey Edwards. Implementation and evaluation of a hypercube-based method for spatiotemporal exploration and analysis. *ISPRS journal of photogrammetry and remote sensing*, 59(1):6–20, 2004.

[95] Stefan Berchtold, Christian Boehm, and Hans-Peter Kriegel. High-dimensional index structure, November 28 2000. US Patent 6,154,746.

[96] Larry S Davis. *Foundations of image understanding*, volume 628. Springer Science & Business Media, 2012.

[97] Hanan Samet. Using quadtrees to represent spatial data. In *Computer Architectures for Spatially Distributed Data*, pages 229–247. Springer, 1985.