Data Structures for the Interactive Visual Analysis of Urban Data

DISSERTATION

Submitted in Partial Fulfillment of

the Requirements for

the Degree of

DOCTOR OF PHILOSOPHY (Computer Science)

at the

NEW YORK UNIVERSITY
TANDON SCHOOL OF ENGINEERING

by

Fabio Miranda

September 2018

Data Structures for the Interactive Visual Analysis of Urban Data

DISSERTATION

Submitted in Partial Fulfillment of

the Requirements for

the Degree of

DOCTOR OF PHILOSOPHY (Computer Science)

at the

NEW YORK UNIVERSITY
TANDON SCHOOL OF ENGINEERING

by

Fabio Miranda

September 2018

Approved:

[illegible]

Department Chair Signature

08/28/2018

Date

University ID:	N15229801
Net ID:	fmm283

Approved by the Guidance Committee:

Major: Computer Science

[illegible]

Cláudio T. Silva
Professor of Computer Science
New York University

8/24/18
Date

[illegible]

Enrico Bertini
Professor of Computer Science
New York University

8/24/18
Date

Juliana Freire

Juliana Freire
Professor of Computer Science
New York University

8/24/18
Date

[illegible]

Huy T. Vo
Assistant Professor of Computer Science
The City University of New York

8/24/18
Date

Vita

Fabio Miranda was born in the city of Belo Horizonte, Brazil in June 1985. He has a B.Sc. in Computer Science from the Federal University of Minas Gerais, and a M.Sc. in Computer Science from Pontifical Catholic University of Rio de Janeiro. While completing his master studies, he worked as a researcher and software engineer developing visualization tools for the oil industry. He started his Ph.D. in September 2012, working in a variety of areas, including large-scale data analytics, data management and data visualization. He has received an award for SIGMOD 2018 Best Demonstration and the Pearl Brownstein Doctoral Research Award. His work has been featured in The New York Times, The Economist and Architectural Digest, among others.

Webpage: `https://fmiranda.me/`

Acknowledgements

This dissertation would not be possible without the unconditional support from my mother, father, wife and family. Even thought my parents felt the pain of seeing a son move to a different state (and then to a different country), they always encouraged me to pursue my goals.

I would like to especially thank my advisor, Cláudio Silva, for the opportunity to join his group at New York University, the continuous support and guidance throughout my Ph.D. studies, and the opportunity to work on incredible projects. I would also like to especially thank the members of my Ph.D. committee, Juliana Freire, Enrico Bertini and Huy T. Vo, for their valuable feedback.

During the course of my Ph.D. studies I had the opportunity to work with incredible researchers that were crucial in my training as a researcher. Among these, I would like to especially thank Harish Doraiswamy, Marcos Lage and Lauro Lins.

I would like to thank Waldemar Celes, from Pontifical Catholic University of Rio de Janeiro, and Luiz Chaimowicz, from Federal University of Minas Gerais, for guiding me through my first steps in my academic life during the M.Sc and B.Sc.

I would also like to express my gratitude to the mentors that I had during the internships: Patricia Crossno, from Sandia National Laboratories, James Klosowski, from AT&T Labs Research, Bruna D'Amora, from IBM Research, and Venkat Vishwanath, from Argonne National Laboratory.

Among the collaborators in the research projects I was part of, I would also like to thank: Juan Pablo Bello, Charlie Mydlarz, Justin Salamon, Yitzchak Lockerman, Luc Wilson, Mondrian Hsieh, Abdullah Kurkcu, Kaan Ozbay, Bruno Gonçalves and Kai Zhao.

I would also like to thank the funding agencies that supported the work presented here: National Science Foundation (NSF awards CCF-1533564, CNS-1229185, CNS-1544753, CNS-1730396), and C2SMART.

Finally, I would like to thank all friends and members of the VIDA Lab for the amazing research environment and incredible moments inside and outside the lab.

Fabio Miranda
September 2018

To my mother and father.

ABSTRACT

**Data Structures for the
Interactive Visual Analysis of Urban Data**

by

Fabio Miranda

Advisor: Prof. Cláudio T. Silva, Ph.D.

**Submitted in Partial Fulfillment of the Requirements for
the Degree of Doctor of Philosophy (Computer Science)**

September 2018

A modern city is the combination of several complex and intertwined systems: transportation, street layouts, public utilities and land use all interact with one another in a process that shapes and forms the city, ultimately influencing how people occupy, move and utilize the many services provided by an urban center. The rapid increase in urbanization in the past century has made this process much more complex, as cities struggle to satisfy the demands imposed by an even larger number of people. It is therefore essential to have a better understanding of different aspects of the city, and how they change over time.

Fortunately, technological innovations have enabled the automatic collection of a diverse set of data that captures the behavior of different components of a city, such as its residents, existing infrastructure and the environment. This creates new opportunities to better understand different dimensions and facets of the city.

By exploring and analyzing urban data through visual analytics systems, domain experts and stakeholders can gain new insights about the data and ultimately the city. However, to be effective, these systems must be *interactive*, requiring sub-second response times in order to maximize data set coverage during analysis, as well as the rate in which users generate hypotheses. General approaches often fail to drive this interactivity, either because they do not efficiently handle the large size of the data or because they were not designed for a specific task. In this dissertation, we present solutions to enable the interactive exploration of different types of large urban data sets.

First, we tackle the challenge of analyzing large temporal data. Advances in technology coupled with the availability of low-cost sensors have resulted in the continuous generation of large time series from several sources. In order to visually explore and compare these time series at different time scales, analysts need to execute online analytical processing (OLAP) queries that include constraints and group-by's at multiple temporal resolutions. To enable interactive OLAP queries over large time series, we propose Time Lattice, a memory-efficient data structure that makes use of the implicit temporal hierarchy; it materializes a subset of a data cube and is designed to handle streaming data.

Second, we consider spatiotemporal data. Social media data usually contains not only a temporal (*when*) and spatial (*where*) dimensions, but also associated keywords describing *what* is the data. A popular way to analyze such data is through the ranking of the top objects within a spatial and temporal selection. For example, a popular way to analyze Twitter social media data is to compute the most popular hashtags in a given region (e.g., neighborhood) and time (e.g., summer). For this, we propose TopKube, a rank-aware data cube that enables the computation of such queries interactively by merging pre-computed ranked lists using an hybrid approach that combines a sweep merging algorithm with Fagin's Threshold Algorithm for Top-k queries.

Next, we explore the challenges of analyzing 3D data. The rise of collaborative mapping initiatives has created data sets that go beyond the usual flatland spatiotemporal data, capturing also the 3D geometry of the city itself. This creates the perfect opportunity to explore *new* urban properties at a scale that was not possible before. One of these properties is the impact of shadows from buildings.

Shadows can potentially infringe on the “right to light” of other citizens in the community through the occlusion of direct sunlight by shading public spaces. Determining its effects requires the accumulation of shadows over time across different periods in a year. For this, we propose Shadow Accrual Maps, a data structure that uses the properties of sun movement to track the changing position of shadows within a fixed time interval in order to efficiently compute shadow accumulation.

Finally, we present the first steps towards visualizing the different views of a city. Using a data set composed of over 40 million street-level images captured by cars in New York City over a period of one year, we create an interactive visualization framework for exploring the various visuals offered by a city. By integrating the images with urban data sets, we allow the interactive exploration of city views under different constraints, such as weather, building construction permits and noise complaints.

We demonstrate the utility of our proposals through a number of case studies set in New York City, highlighting its usefulness in the study of common urban problems, such as noise and shadow impact on public spaces.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cities have been at the center of human organization for more than three thousand years [1]. This concentration of people with different backgrounds and skills living in close quarters plays a fundamental role in the creation of culture, innovation and economical growth [2]. Today, more than ever, a city is a combination of several complex and intertwined systems: transportation systems, housing, street layouts, utilities and land use all interact with one another in a process that shapes and forms the city, ultimately influencing how people occupy, move and utilize the many services provided by an urban center. The rapid increase in urbanization in the past century [3] has made this process much more complex, as cities struggle to satisfy the demands imposed by an ever larger number of people.

The new problems created by urban growth gave rise to the necessity to better define and understand the city, its problems and its dwellers. An early and iconic case of such necessity is the Broad Street cholera outbreak. In 1854, the district of Soho in London suffered from a growing population and a lack of proper sanitary conditions, ultimately leading to a cholera outbreak that killed 616 people. The prevailing theory was that the disease was transmitted by air. The work by epidemiologist John Snow in mapping and analyzing the spatial distribution of cholera cases (see Figure 1.1) was fundamental in identifying the actual cause of cholera: contaminated water [4].

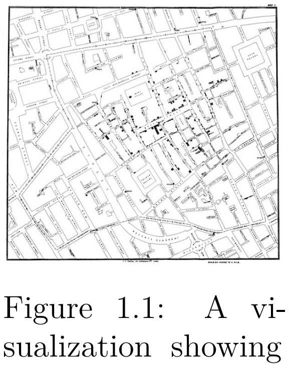


Figure 1.1: A visualization showing the distribution of cholera cases around Broad Street in 1854.

In the begging of the 20th century, urban sociologists also explored the urban challenges, with studies from Max Weber on the definition of a city [5] and Louis Wirth on what constitutes an urban way of life [6]. The seminal work by Ernest Burgess and Robert E. Park was one of the first to use data to understand the city and to propose a model of urban organization [7]. Wirth, Burgess, Park and others were among the researchers of what would be named the *Chicago school*, a group of urban sociologists at the University of Chicago that proposed to collect and analyze urban-related data in the 1920's and 1930's [8, 9].

The importance of data only grew in the decades following the *Chicago school*. Urban analyses, however, were limited to a very coarse and manually collected set of data, that in most cases did not capture the dynamism of modern cities [10]. For instance, a popular source for urban analyses was the census data, which, in the United States, is performed only every ten years.

In the past decade, technological innovations have enabled the automatic collection of a diverse set of data that captures the behavior of the different components of a city, namely its residents, existing infrastructure, the environment, and interactions between these elements. As a result of this, the term *smart city* has been coined to describe cities that use technology as a way to make its infrastructure components and services more efficient [11, 12]. Initiatives such as open data portals maintained by city agencies have democratized the access to data, and allowed different stakeholders to engage in urban analyses [13, 14, 15, 16, 17]. Private companies are also increasingly making their data available, a sign that more entities understand the power of open data and the potential that can be unlocked by combining diverse sources of data [18, 19].

Such recent initiatives create new opportunities to better understand different *dimensions* and *facets* of the city, and how they change over the course of space and time. However, urban data analysis has often been limited to well-defined questions, or *confirmatory data analysis* [20, 21]. In this analysis, a domain expert usually formulates an hypothesis, a data scientist performs the analyses and finally the expert inspects the results to either verify or disprove the hypothesis. Given the complexity and sheer volume of urban data being generate, this process cannot scale. Making an analyst wait too long for the result of its analysis might mean breaking her flow of though, and be the difference between capturing an idea or not.

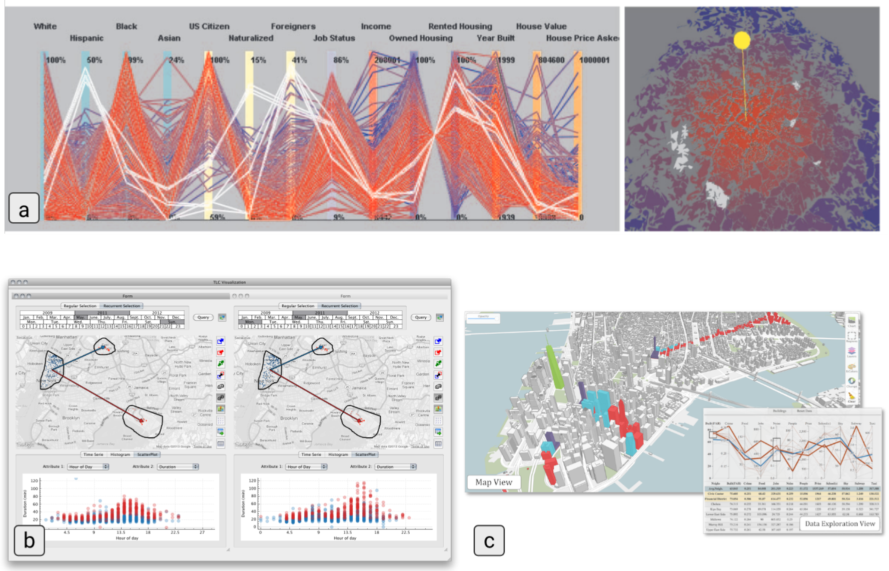


Figure 1.2: Examples of visual analytics tools. **(a)** Legible Cities: visualization of census data [22]. **(b)** TaxiVis: visualization of New York City taxi trips [23]. **(c)** Urbane: visualization of neighborhood-level data and view impact analysis [24].

Not surprisingly, there have been several tools and techniques with the goal of allowing for interactive exploration of urban data sets (such as the ones in Figure 1.2 and [25, 26, 27, 28, 29]). However, there are several challenges in driving and enabling interactivity in such tools. First, it is important to understand the queries that are most useful in the analysis process of a given data. Second, since urban data is usually large and complex, it is necessary to design structures that are able to solve the posed query in a small amount of time. In fact, the importance of interactivity and low latency during analysis tasks has been studied by Liu and Heer [30]. They conclude that even a half second difference between user input and visual display can significantly impact visual analysis, reducing interaction and data set coverage, as well as the rate in which users make observations, draw generalizations and generate hypotheses. It is therefore important to have frameworks and structures that are able to drive interactive visual analysis, specifically tailored to the task at hand. Such systems create new opportunities for city governments and domain experts to engage in data-driven science to better understand cities, and ultimately improve the lives of their residents.

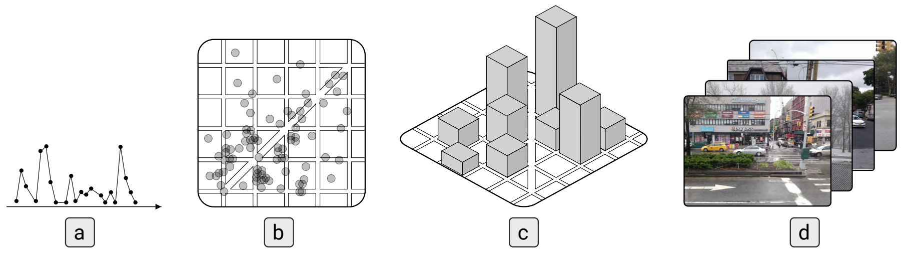


Figure 1.3: Overview of types of data explored in this dissertation. **(a)** Time series from stationary sensors. **(b)** Spatio-temporal data from social media. **(c)** 3D geometry of the city. **(d)** Street-level images.

1.1 Contributions

The growth in the availability of data sets describing different urban phenomena has create an opportunity to better understand a city and its processes. A city is an amalgam of systems that interact at different spatial resolutions and at different temporal scales, and this complexity is reflected in urban data, which often comes in diverse and complex data types (see Figure 1.3). For instance, data from stationary sensors is commonly one dimensional, while data describing the buildings and infrastructure of the city is three dimensional. On top of that, the analyses tasks often required when exploring urban data are computationally expensive, making the requirement of interactivity even harder to be achieved. General approaches often fail to drive this interactivity, either because they do not efficiently handle the large size of the data or because they were not designed for a specific task. In this dissertation, we focus our efforts on enabling interactive visual analyses of applications that explore different types of data, each covering a particular aspect of the city. Ultimately, we want to enable a deeper understanding of the different *dimensions* and *facets* that make a city [12]. Below, we summarize the contributions of each chapter of this dissertation.

Analyzing Temporal Data. The massive deployment of sensors throughout a city results in the continuous stream of data as time series from up to thousands of objects. The exploration of such data typically involves slicing and dicing the time series over different temporal resolutions together with multiple constraints.

We propose the Time Lattice data structure [31], that makes use of the implicit temporal hierarchy to enable interactive OLAP queries over large time series.

Our proposal is a subset of a fully materialized cube and is designed to handle fast updates and streaming data. The structure is used to drive the interactivity of Noise Profiler, a tool used to visually analyze the time series data generated by more than 48 sensors deployed around New York City and that have been collecting noise levels for more than a year at a high resolution. With this, we are able to interactively compute and compare noise baselines across several temporal dimensions of sensors deployed in different parts of the city.

Analyzing Spatiotemporal Social Media Data. The widespread use of GPS-enabled devices has created a large volume of geotagged data. Common to social media data, these devices attach not only a temporal dimension (*when*) but also a spatial one (*where*) to any user-generated content. On top of that, it is common to have several extra fields in any data generated by the user describing *what* is the data. A popular way to analyze such data is through the ranking of the top objects incident to an arbitrary spatial and temporal selection. For instance, a popular way to analyze Twitter social media data is to compute the most popular hashtags in a given region (e.g., neighborhood, city) and time (e.g., Christmas).

We propose TopKube [32], a rank-aware data structure that merges pre-computed ranked lists considering spatial and temporal constraints. The structure is used as the backend of a visual analysis tool used to interactively explore a variety of social media data. Such tool can be used to gain new insights about the activity and interests of people across neighborhoods.

Analyzing the 3D Geometry of the City. The rise of collaborative mapping initiatives has created data sets that go beyond the usual flatland spatiotemporal data. The OpenStreetMap project, for instance, makes available highly detailed 3D geometry information about the buildings of several cities [33, 34, 35]. This creates the perfect opportunity to create and explore *new* urban properties at a scale that was not possible before. One of these properties is the impact of shadows from buildings. Shadows can potentially infringe on the "right to light" of other citizens in the community through the occlusion of direct sunlight by shading public spaces. This can inhibit vegetation growth and reduce solar energy potential. On the other hand, shadows can also be beneficial by reducing the urban heat island effect, or by providing a comfortable environment for park goers. Determining the effects of shadows requires the accumulation of shadows over time across different periods.

We propose Shadow Accrual Maps [36], a data structure that uses the properties of sun movement to track the changing position of shadows within a fixed time interval in order to efficiently compute shadow accumulation. The structure is used to develop an interactive visual analysis system *Shadow Profiler*, targeted at city planners and architects that allows them to test the impact of shadows for different development scenarios.

Exploring the Visuals of a City. The previous applications focus on understanding the city through the measuring and quantification of a subset of attributes from interesting urban data. These analyses, however, fail to capture the visual experiences of city dwellers: how does a city look during early mornings or snowy afternoons, across different neighborhoods? By using a data set composed of over 40 million images captured by cars in New York City over a period of one year, we take the first steps in creating a visualization framework for exploring the various visuals offered by a city. By integrating the images with other urban data sets, we allow for the interactive exploration of city views under different constraints, such as weather.

1.2 Outline

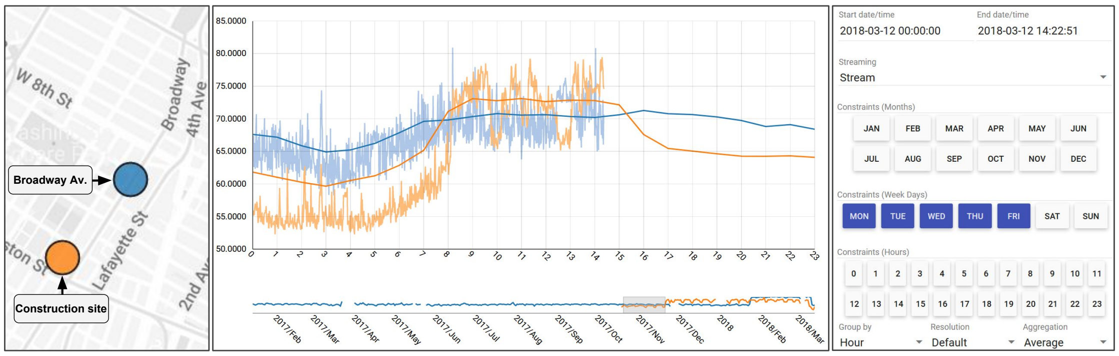
Chapter 2 presents the Time Lattice data structure for the interactive visual analysis of large time series. Chapter 3 describes the TopKube rank-aware data cube for the real-time ranking of geotagged keywords, such as Tweets. Chapter 4 presents Shadow Accrual Maps, for the efficient accumulation of city-scale shadows over time. Chapter 5 presents a system for the exploration of a large collection of street-level images. Finally, Chapter 6 concludes the dissertation, highlighting potential future works.

Chapter 2

Temporal Data: Interactive Visual Analysis of Large Time Series

With the massive adoption of the *Internet of Things* (IoT) in various scenarios ranging from smart home devices and smart cities to medical and healthcare applications, interactive visualization frameworks are becoming paramount in the exploration and analysis of the data generated by these systems. Any such IoT setup continuously transmits data as a time series from tens and hundreds up to thousands of objects (or sensors). The exploration of these data typically requires complex online analytical processing (OLAP) queries that involve slicing and dicing the time series over different temporal resolutions together with multiple constraints and custom aggregations. The use of a visual interface adds an additional constraint: the queries must be *interactive*, since high latency queries can break the flow of thought, making it difficult for the user to effectively make observations and generate hypotheses [30].

In this chapter, we are specifically interested in the analysis of time series from acoustic sensors deployed to help map and understand the noisescape in cities. Noise is an ever present issue in urban environments. Besides being an annoyance, noise can have a negative effect on education and overall health [37]. To combat these problems, cities have developed noise codes to regulate activities that tend to produce sounds (see e.g., [38, 39]). To help monitor noise levels in New York City,



Start date/time 2018-03-12 00:00:00

End date/time 2018-03-12 14:22:51

Streaming: Stream

Group by: Hour

Resolution: Default

Aggregation: Average

Figure 2.1: Using Noise Profiler to analyze OLAP queries over acoustic data from sensors deployed in New York City. A group-by hour is used as a baseline for ambient noise (smooth line), highlighting the difference between the noise profile of two locations during weekdays. One sensor (blue) is close to a main road (Broadway Av.) and has a constant dB_A level throughout the hours of the day; the other sensor (orange) is close to a major construction site and has a distinctly higher dB_A level during construction hours between 7 a.m. and 5 p.m. The live streaming data (fluctuating line) can be used to get instantaneous information about the noise level captured by the sensors, and inform city agency noise enforcement teams about possible noise code violations such as construction sites operating outside of their allotted construction hours.

as well as to aid government agencies in regulating noise throughout the city, researchers part of the *Sounds of New York City* (SONYC) project have developed and deployed low cost sensors that have the ability to measure and stream accurate sound pressure level (SPL) decibel data at high temporal resolutions (typically every second) [40, 41, 42]. Thirty six such sensors have been collecting data for over a year, in addition to another twelve new sensors that have been deployed since. As the size of this network continues to grow, the amount of data produced by the sensors becomes virtually unbounded.

This necessitates the ability to handle analysis queries efficiently on such large time series data, in particular, the more complex OLAP queries that require aggregations of the data across multiple temporal resolutions. For example, noise enforcement agencies can assess a breach if the noise level is greater than the ambient background noise. However, the ambient background noise patterns are spatially localized and vary depending on the time (e.g., peak hours, night time, weekdays, weekends, etc.). So, to identify these patterns over weekdays, as shown in Figure 2.1, the following query is issued using a visual interface over data from sensors present in the different regions of interest:

select *time series* ***during*** weekdays ***groupby*** *hour*

Furthermore, not only can the user restrict the time range over which to perform the above query (e.g., in Figure 2.1, the time range is from October 2017 to December 2017), but depending on the location and its conditions (e.g., tourist spots), more constraints might also be *interactively* added to this query. Since users can continuously alter the constraints through the visual interface, it is crucial that these queries have low latency to enable seamless interaction.

Problem Statement and Challenges. The goal of this work is to design a time series data structure that supports OLAP queries and has the following important properties:

1. Interactive queries;
2. Interactive updates from new data; and
3. Low memory overhead.

Two common approaches to support OLAP queries are to use either database systems catered for time series, or data cube-based solutions. However, neither of the approaches satisfy all of the above requirements that are crucial for real-time visual analysis of the data.

Traditional time series databases [43, 44, 45, 46], by supporting the powerful SQL-like syntax, can execute a wide range of queries including the OLAP queries with temporal constraints that are of interest in this work. They are often memory efficient, and support updates over new data. To execute a given query, these systems typically use an index to first retrieve intermediate results based on the constraints. The query results are then computed by explicitly aggregating the intermediate results. Unfortunately, such strategy fails to be interactive when handling data at the scale that is now available (see Section 2.4).

Data cube-based structures [32, 47, 48, 49], on the other hand, have extremely low latency to OLAP queries. However, the size of these data structures increases exponentially with the number of dimensions. In case of a time series, the dimensions correspond to the discrete temporal resolutions for a time series. Moreover, to support temporal constraints in these queries, the time resolution of these constraints should also be a dimension of the cube. For example, specifying the time period of interest with an accuracy up to a minute requires *minute* to be a dimension of the data cube. This further increases the space overhead. While this might be admissible when working with a single time series, it becomes impractical when working with several tens to hundreds of time series that is now commonplace with IoT systems. Additionally, the more practical memory-optimized data cube structures [47, 48] do not support updates with new (or streaming) data, thus requiring the re-computation of the entire structure every time. Given that the cube creation time can take minutes even for reasonably small data sizes, this approach becomes impractical for handling multiple large streaming time series data.

Contributions. In this chapter, we present a new data structure, *Time Lattice*, that can perform OLAP queries over time series at interactive rates. The key idea in its design is to make use of the implicit hierarchy present in temporal resolutions to materialize a sub-lattice of the data cube. This helps avoid the curse of dimensionality common with other cube-based structures and results in a *linear memory overhead*, while still being able to conceptually represent the entire cube. This drastic reduction in memory also allows us to augment our data structure with additional summaries, thus supporting the computation of measures that are otherwise not easily supported. More importantly, unlike existing approaches, our data structure allows *constant amortized time updates.*

To demonstrate the effectiveness of Time Lattice, we develop *Noise Profiler*, a proof of concept web-based visualization system, that is being used in the SONYC project to analyze acoustic data from New York City.

To summarize, our contributions are as follows:

- We introduce *Time Lattice*, a data structure that supports multi-resolution OLAP queries on time series at interactive rates. It has a *linear memory overhead*, and supports *constant amortized time updates* with new data.
- We show experimental results demonstrating both the time as well as space efficiency of Time Lattice.
- We develop Noise Profiler, a web-based visualization system to *simultaneously analyze* multiple streams of data generated from the SONYC sensors. Note that,

without the underlying efficient data structure, it would not be possible to visually analyze such multiple streams in real time.

- We demonstrate the utility of Time Lattice through a set of case studies performed by subject matter experts, and which are of interest to the end users of the SONYC project.

2.1 Related Work

Time Series Databases. Several databases have been proposed to facilitate data acquisition and data querying of time stamped data. Their architecture and design vary greatly depending on their goal. One class of database systems such as tsdb [50], Respawn [51] and Gorilla [43] are primarily concerned with providing the user with monitoring capabilities, and lack support for complex analytical queries. Respawn [51] proposes a multi-resolution time series data store to efficiently execute range queries. While it efficiently speedup range queries, it does not support aggregations (such as group-by's) over any temporal resolution.

One of the most popular database to support analytical queries on time series is InfluxDB [45], which offers a SQL-like language for queries, including *rollups* and *drilldowns*. KairosDB [46] is another popular time series database that uses Apache Cassandra for data storage, and provides much of the same features as InfluxDB. Timescale [52], on the other hand, builds on top of the popular Postgres to offer a database solution tailored for time series. As we show later in Section 2.4, a major drawback of these solutions is that they cannot drive interactive visualization, with complex OLAP queries requiring several seconds to execute. For a more detailed survey on existing time series data management systems, we refer the reader to the following surveys by Jensen et al. [53] and Bader et al. [44].

Data Cube. Data cube [54] is a popular method designed specifically to handle OLAP analytical queries. It pre-computes aggregations over every possible combination of dimensions of a data set in order to support low-latency queries. It has been extended to support data sets from different domains, such as graphs [55] and text [56]. The main drawback of a data cube is the exponential growth of the cube with increasing dimensions making them impractical when working with large data sets. A common approach to reduce the size of a data cube is to materialize only a subset of all possible dimension combinations. One such approach, called iceberg cube [57], only stores aggregations that satisfy a given condition (specified as a threshold), and discards any values not above this threshold. While this approach is suitable for the analysis of historical data, updates become unfeasible since new data dynamically changes the aggregation requiring access to previously discarded values.

More recently, with the focus on spatio-temporal data, several approaches have been proposed to deal with the curse of dimensionality. Nanocube [47] uses shared links to avoid unnecessary data replication along the data cube. However, the above memory reduction scheme is not sufficient to reduce the structure size when considering high resolution, dense time series typically available from IoT devices (see Section 2.4). Hashedcube [48], on the other hand, uses pivots to efficiently compute a subset of the aggregations on the fly from the raw data, rather than pre-computing all of them, thus achieving a considerably lower memory footprint. To do this, it requires the data to be sorted according to its dimensions. While both nanocube and hashedcube support low latency queries capable of driving interactive visualizations, they cannot handle data updates. Han et al. [58] tackle the memory explosion by restricting the analysis to a temporal window. This is accomplished by a data cube that, while updating new data points, discards old points (and the corresponding aggregations) based on a user defined retention policy. A similar retention approach is also used by Duan et al. [59]. While this approach is suitable for monitoring applications requiring analysis on recent history, it relies on approximate queries and cannot be used for historical analysis.

Our goal is have a data structure that supports real-time queries for both historical analysis as well as monitoring applications, while still being memory efficient. To accomplish this, we choose a materialization of the data cube based on the intrinsic temporal hierarchy that enables constant amortized time updates, as well as real-time query execution. However, note that the proposed data structure is not a replacement for general data cubes, which are structures applicable to any data set. Rather, it provides an efficient alternative when working with large time series and OLAP queries that slice and dice the time series over the temporal resolutions.

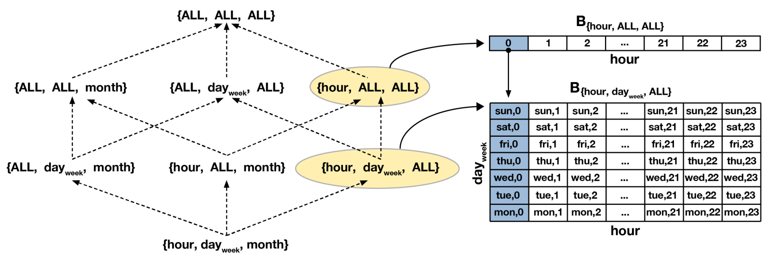

{ALL, ALL, ALL}
{ALL, ALL, month}
{ALL, day_week, ALL}
{hour, ALL, ALL}
{ALL, day_week, month}
{hour, ALL, month}
{hour, day_week, ALL}
{hour, day_week, month}
B{hour, ALL, ALL}
0 1 2 ... 21 22 23
hour
B{hour, day_week, ALL}
sun,0 sun,1 sun,2 ... sun,21 sun,22 sun,23
sat,0 sat,1 sat,2 ... sat,21 sat,22 sat,23
fri,0 fri,1 fri,2 ... fri,21 fri,22 fri,23
thu,0 thu,1 thu,2 ... thu,21 thu,22 thu,23
wed,0 wed,1 wed,2 ... wed,21 wed,22 wed,23
tue,0 tue,1 tue,2 ... tue,21 tue,22 tue,23
mon,0 mon,1 mon,2 ... mon,21 mon,22 mon,23
day_week
hour

Figure 2.2: Data cube with $D = \{hour, day_{week}, month\}$ has a total of $2^{|D|}$ cuboids, where each cuboid stores the aggregations for all possible values of its dimensions.

Time Series Visualization. Time stamped data has long been studied and visualized in multiple domains. Several studies propose different metaphors and interactions when dealing with time series, such as applying lenses [60], clustering values into calendar-based bins [61] or re-ordering of the series at different aggregations to allow for an easier exploration [62]. The perception impact on the visualization of multiple time series has been studied by Javed et al. [63]. A full survey of different techniques was presented by Silva and Catarci [64], Müller and Schumann [65] and Aigner et al. [66]. Note that all of these approaches are orthogonal to this work. While their goal is to provide new visual metaphors, ours is to support real-time execution of queries that are used to generate the required visualizations. The visualization of time series in multiple resolutions has also been a topic of study. Berry and Munzner [67] aggregate the data into bins prior to the visualization. Hao et al. [68] proposed a distortion technique that generates visualizations where more visual space is allocated to data according to a measurement of interest. Jugel et al. [69] proposed M4, a technique to aggregate and reduce time series considering screen space properties. All of these approaches, however, do not focus on OLAP-type queries, limiting their techniques to essentially a range query at a coarser resolution.

Another popular area of research associated with time series is the querying of similar patterns in a time series [70, 71, 72]. Time Lattice can augment these approaches by speeding up sub-queries that are commonly used by them.

2.2 Time Lattice

The primary goal of this work is to efficiently execute queries of the following type over an input time series:

$$\begin{aligned}\textbf{\textit{select}}\ \textit{time series}\ \textbf{\textit{between}}\ t_1\ \textit{and}\ t_2 \\ \textbf{\textit{where}}\ \textit{constraints}\ C \\ \textbf{\textit{groupby}}\ \textit{resolutions}\ G\end{aligned}$$

where, t_1 and t_2 specify the time period of the data to consider. The constraints $C = \bigcup_r \{C_r\}$ define the constraints over each temporal resolution r. Here, C_r specifies a set of values in resolution r that have to be satisfied. The resolutions $g \in G$ specify the resolutions on which to perform the *group-by*. For example, if the query in Section 2 has to be executed only for data from the last 6 months of 2017, we set $t_1 =$ 2017-06-01T00:00; $t_2 =$ 2017-11-30T:23:59; $C = \Big\{C_{day_{week}} = \{Monday, ..., Friday\}\Big\}$; and $G = \{hour\}$.

In this section, we describe the main data structure, Time Lattice, and discuss its properties. We also explain the query execution strategy using Time Lattice and describe extensions to the data structure that enable additional features such as support for join queries and multiple aggregations.

2.2.1 Data Structure

A *data cube* [54] is a method that was designed to efficiently answer aggregate queries such as the one shown above. Here, the resolutions of time are modeled as the *dimensions* D of a data cube. However, unlike general data sets, the dimensions of time corresponding to the different temporal resolutions are hierarchically dependent. We make use of this property to design a data structure that is both memory efficient and supports interactive aggregate queries. To avoid the exponential memory overhead of a data cube, we compute only a subset of the data cube. We then make use of the inter-dependency between the temporal resolutions to efficiently compute on-the-fly query results. In this section, we first provide a brief overview of data cubes followed by describing in detail the proposed data structure. We use the terms resolution and dimensions interchangeably in the remainder of the text.

Table 2.1: List of symbols.

T	Discrete space representing time.
$f : T \to \mathbb{R}$	Time series.
D	Dimensions of the data cube. It corresponds to the temporal resolutions in case of a time series.
$\mathcal{P}(D)$	Power set of D.
$\prec$	Partial order defined on the temporal resolutions.
H	Hasse diagram of the poset $(D, \prec)$.
B_r	Cuboid corresponding to resolution r.
$\alpha_r(t)$	Association function mapping time step t to an offset in B_r.
$\pi_{r \to r'}(i)$	Containment function mapping an element in B_r to an element in $B_{r'}$, where $r \to r' \in H$.

Preliminaries: Data Cubes. Consider a time series $f : T \to \mathbb{R}$, which maps each time step of a discrete temporal space T to a real value. Without loss of generality, let the resolution of T be seconds and be represented using epoch time (i.e., seconds since January 1, 1970, Midnight UTC). Let f be defined for every second within a time interval $[t_1, t_2)$, $t_1, t_2 \in T$. For ease of exposition, assume that there are no gaps in the time series, that is, the function f is defined for all $t_1 \leq t < t_2$. Since we are working with time, f can also be analyzed in resolutions coarser than a second, such as *minute*, *hour*, *day of week* (day_{week}), etc.

A data cube represents all possible aggregations over the dimensions in D. Formally, a data cube represents the 2^d *cuboids* corresponding to the elements of the power set $\mathcal{P}(D)$, where $d = |D|$. For example, given dimensions $D = \{hour, day_{week}, month\}$:

$$\begin{aligned}\mathcal{P}(D) = \Big\{&\emptyset, \{hour\}, \{day_{week}\}, \{month\}, \{hour, day_{week}\},\\ &\{day_{week}, month\}, \{hour, month\},\\ &\{hour, day_{week}, month\}\Big\}\end{aligned}$$

The set of cuboids for the data cube in the above example is shown in Figure 2.2.

The *dimension* of a cuboid B_P, $P \in \mathcal{P}(D)$, is equal to $|P|$. For example, the element $\{day_{week}\}$ forms a 1-dimensional cuboid while $\{hour, day_{week}\}$ forms a 2-dimensional cuboid. A k-dimensional cuboid (or k-cuboid) stores all possible aggregations corresponding to its k dimensions. For example, the 2-cuboid $\{hour, day_{week}, ALL\}$, corresponding to the element $\{hour, day_{week}\} \in \mathcal{P}(D)$, stores the aggregations for all possible ($hour$, day_{week}) values. Here, the aggregation is performed over the other $d-k$ dimensions represented by ALL, which in the above example is *month*. Thus, this cuboid has size 24×7 (there are 24 possible hours and 7 possible days). In general, the size of a k-cuboid, i.e., the number of aggregations stored by the cuboid, is equal to the product of the cardinality of each of its dimensions.

A fully materialized data cube pre-computes and stores all 2^d cuboids corresponding to $\mathcal{P}(D)$. As a rule of thumb, the number of dimensions to use to create a cube depends on the resolution of the constraints used in the query. In the above example, to support queries that group by or filter over arbitrary time ranges specified in the resolution of minutes, the dimension *minutes* should be added to D. When a new dimension is added, not only does the number of cuboids increases by a factor of 2 (2^3 to 2^4 in the example), but the total number of aggregations stored (corresponding to all the cuboids) increases by a factor equal to the number of categories in that dimension (60 in case the dimension minute is added to D, since there are 60 possible values denoting a minute). Clearly, the size of the data cube increases exponentially with new dimensions, and can quickly become intractable when working with resolutions commonly used in time series analyses.

The Time Lattice structure. Instead of materializing the entire data cube, we use the intrinsic hierarchy present in time to materialize only a subset of this cube. Formally, let $D = \{r_1, r_2, \ldots, r_d\}$ denote the different temporal resolutions. Let $\prec$ denote a partial order defined on D, such that $r_i \prec r_j$ if the time stamps in resolution r_i can be partitioned based on the time stamps in resolution r_j. For example, $minute \prec hour$ and $hour \prec day_{week}$, since the time stamps specified in minutes can be partitioned based on the hour of that time stamp, and similarly hours can be partitioned by days. Note that the above partial order is different from the partial order that defines the data cube itself (defined by the inclusion function [54]). Let H denote the *Hasse diagram* of the partially ordered set, or *poset*,

$(D, \prec)$. The nodes of H correspond to the dimensions in D, and an edge exists from r_i to r_j if r_j *covers* r_i, i.e., $r_i \prec r_j$ and $\nexists r_k | r_i \prec r_k \prec r_j$. In the above example, even though $minute \prec day_{week}$, this edge does not exist in H since day_{week} does not cover $minute$ ($\exists hour$ s.t. $minute \prec hour \prec day_{week}$). Figure 2.3 shows the Hasse diagram for the poset covering common temporal resolutions used in this work.

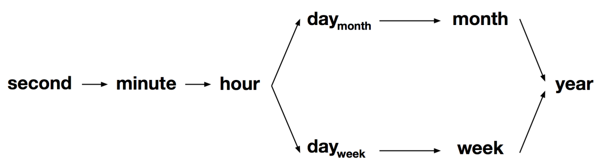


Figure 2.3: Hasse diagram denoting the poset defined on the temporal resolutions used in this work.

Time Lattice materializes cuboids using H as follows. Consider a maximal path $(r_{i_1}, r_{i_2}, \ldots, r, \ldots, r_{i_n})$ in H such that $r_{i_1} \prec r_{i_2} \prec \ldots \prec r \prec \ldots \prec r_{i_n}$. The cuboid $(ALL, ALL, \ldots, ALL, r, \ldots, r_{i_n})$ is materialized corresponding to the node r in this path. For example, consider the node day_{month} in the poset defined in Figure 2.3. This results in materializing the cuboid $(ALL, ALL, ALL, day_{month}, Month, Year)$. Next, consider a resolution r which is not part of this path. A maximal path in H that includes r is next chosen to be materialized. This process is repeated until there is at least one cuboid corresponding all resolutions in H. The Time Lattice is the union of all the cuboids resulting from the above materialization. Note that, since each cuboid B_P that is materialized corresponds to a resolution $r \in H$, we refer to this cuboid using r as B_r.

Such a materialization has several advantages:

- Each materialized cuboid B_r can be represented by a contiguous array such that the aggregate values stored in B_r follow a chronological order representing a continuous time series in resolution r. Thus, the Time Lattice can have a simple array-based implementation.
- Consider the resolutions day_{month} and day_{week}. Even though they are conceptually different (the categories have different range: $\{1, 2, 3, \ldots\}$ vs. $\{Mon, Tue, \ldots\}$), the individual array elements of $B_{day_{month}}$ and $B_{day_{week}}$ correspond to the same days. Thus, the same array can be shared by both these cuboids.
- Because of the chronological ordering, the different cuboids can be *implicitly indexed* based on the resolution r. This implicit index is formally defined using

Table 2.2: Association functions. Here $y()$ and $m()$ return the year and month respectively for a given time stamp, and $weeksbetween()$ returns then number of weeks between two time stamps.

$\alpha_{second}(t)$	$B_{second}[t - t_1]$
$\alpha_{minute}(t)$	$B_{minute}[\lfloor \frac{t}{60} - \lfloor \frac{t_1}{60} \rfloor \rfloor]$
$\alpha_{hour}(t)$	$B_{hour}[\lfloor \frac{t}{60*60} - \lfloor \frac{t_1}{60*60} \rfloor \rfloor]$
$\alpha_{day}(t)$	$B_{day}[\lfloor \frac{t}{24*60*60} - \lfloor \frac{t_1}{24*60*60} \rfloor \rfloor]$
$\alpha_{week}(t)$	$B_{week}[weeksbetween(t, t_1)]$
$\alpha_{month}(t)$	$B_{month}[12 * (y(t) - y(t_1)) + m(t) - m(t_1)]$
$\alpha_{year}(t)$	$B_{year}[y(t) - y(t_1)]$

an *association function*, $\alpha_r(t)$, corresponding to each $r \in H$, which maps a time stamp t to an offset i of the array B_r. That is, $B_r[i]$ stores the aggregated value corresponding to time step t in the cuboid B_r. Table 2.2 lists the association functions α_r used for the resolutions in Figure 2.3.

- Enables efficient updates to the data structure (see details below).
- The temporal hierarchy also allows for an implicit mapping between array elements across resolutions, enabling efficient "rollups" and "drilldown" operations that are performed on a cube (see Section 2.2.2). This mapping is formally defined by the *containment function* $\pi_{r \to r'}$ which maps an array offset i in resolution r to an offset j in resolution r', whenever there is an edge from r to r' in H. Essentially, $\pi_{r \to r'}(i) = j$ if and only if there exists t such that $\alpha_r(t) = i$ and $\alpha_{r'}(t) = j$. This function can also be parametrically computed similar to the association function. Since π is a many-one function, the inverse mapping $\pi^{-1}_{r \to r'}$ maps an offset in the coarser resolution r' to a sub-array in B_r. This mapping to a sub-array is only possible because of the above mentioned ordering of B_r.
- Helps efficiently execute queries with range constraints as well—only the sub-array(s) within the offsets corresponding to the query range has to be considered.

The elements of the cuboid B_r (i.e., $B_r[i]$) store one or more measurements $\mu_r(i)$. Here, μ can be any distributive and algebraic operation. In our implementation, we store the following distributive aggregates—*minimum*, *maximum*, *sum*, and *count*. This can in turn be used to compute other algebraic aggregates such as *average* (see Section 2.2.3 for more details). Note that if the dimension is the same as the resolution of the underlying time series, then μ simply corresponds to the time series itself.

Space Requirements. Let the size of the time series be n. For analysis purposes, first consider a maximal path $r_1, r_2, \ldots, r_k$ in H s.t. $r_1 \prec r_2 \prec \ldots r_k$. Without loss of generality, let r_1 be the original resolution of the time series. Thus, the B_{r_1} simply corresponds to the underlying data itself. Let the space required for materializing at resolution r_i (size of the array B_{r_i}) be s_i. Therefore, $s_1 = n$. Then, the space required for materializing all arrays (i.e., not counting the base array, which is the underlying time series) is $s = \sum_{i=2}^{k} s_i$. The size s_{i+1} is a fraction of s_i defined by $s_{i+1} = \lceil s_i / a_{i+1} \rceil$, where $a_{i+1} = |\pi^{-1}_{r_i \to r_{i+1}}|$. For example, $a_{minute} = 60$ (60 seconds make a minute), and $a_{day} = 24$ (24 hours make a day). Therefore,

$$\begin{aligned} s &= \sum_{i=2}^{k} s_i \\ &= \frac{s_1}{a_2} + \frac{s_2}{a_3} + \frac{s_3}{a_4} + \ldots + \frac{s_{k-1}}{a_k} \\ &\leq s_1 \times \left(\frac{1}{a_2} + \frac{1}{a_2 \cdot a_3} + \frac{1}{a_2 \cdot a_3 \cdot a_4} + \ldots + \frac{1}{\prod_{i=2}^{k} a_d} \right) + k \\ &\leq s_1 + k \\ &\approx n \ \left\{ \text{assuming } k \ll n \right\} \end{aligned}$$

Let the total number of maximal paths used to materialize the Time Lattice be m. Then, the size of the Time Lattice data structure is bounded by $O(m \cdot n)$. Given that typically m is a very small integer—$m = 2$ for the Hasse diagram in Figure 2.3, the size of the data structure is *linear* in the size of the underlying data. We would like to note that this is not a tight bound. In fact, as we show later in the experiments, the space required by the structure is significantly smaller in practice ($< 2\%$ of n as shown in Section 2.4.2).

Updating the Data Structure. One of the main goals of our proposed data structure is to support updates over new (or streaming) data. Consider an existing Time Lattice structure, and an incoming value of the time series. Since this value will have a time stamp t at the finest resolution (second for the purpose of this work), it will simply be appended to B_{second}. For resolutions $r | second \prec r$, we first need to check if the corresponding array element $B_R[\alpha_r(t)]$ already exists. If it

```
1: function DrillDown($B', R, r, C, G, t_1, t_2$)
2:   $result \leftarrow []$
3:   $B \leftarrow B' \cap B_{R[r]}[\alpha_r(t_1), \alpha_r(t_2)]$
4:   $C_r \leftarrow$ Constraints at resolution $R[r]$
5:   if $|G| = 0$ and $|C| = 0$ then
6:     $result \leftarrow B$
7:   else if $|G| > 0$ or $|C| > 0$ then
8:     $G = G \setminus \{R[r]\}$
9:     $C \leftarrow C \setminus C_r$
10:    for all $b \in B$ do
11:      if $b$ satisfies $C_r$
12:        $result \leftarrow result \cup$
               $\{$DrillDown$(\pi^{-1}_{R[r+1]\rightarrow R[r]}(b), R, r+1, C, G, t_1, t_2)\}$
13:    if $r \in G$ then
14:      $result \leftarrow$ GroupBy$(result, R[r])$
15:  return $result$
16: function Query($C, G, t_1, t_2$)
17:  $r' \leftarrow$ finest resolution in $C \cup G$
18:  $R[] \leftarrow \{$path in $H$ from $r'$ to $year$ containing $C \cup G\}$ $s.t.$ $R[i+1] \prec R[i]$
19:  DrillDown$(B_{year}, R, 0, C, G, t_1, t_2)$
```

Figure 2.4: Pseudo-code for the aggregate query.

does, we need to update the value of the aggregation μ_r to take into account $f(t)$. If this element does not exist, it is first created and appended to B_r and the value of μ_r is appropriately initialized using $f(t)$.

Assuming that the data structure is updated every second, the time complexity becomes $O(d)$ per update, where d is the number of arrays maintained and is bounded by the number of resolutions in H. Oftentimes, it is not critical to have such a high update frequency. For example, instead of updating the structure every second, it would suffice in practice to update it every minute. Let this update be performed every k seconds. In this case, there will be k appends to B_{second}, $\lceil \frac{k}{a_{minute}} \rceil$ updates / appends to B_{minute}, and so on. Thus, when $k \geq d$ (e.g., for a minute-wise update, $k = 60 > d = 7$) the time complexity is $O(k+d)$ for effectively k updates, or $O(\frac{k+d}{k}) = O(1)$ *amortized time* per update.

2.2.2 Querying

Aggregate Query. Aggregate queries (or OLAP-type queries) are primarily used for a more nuanced analysis on the time series data. The algorithm to execute such a

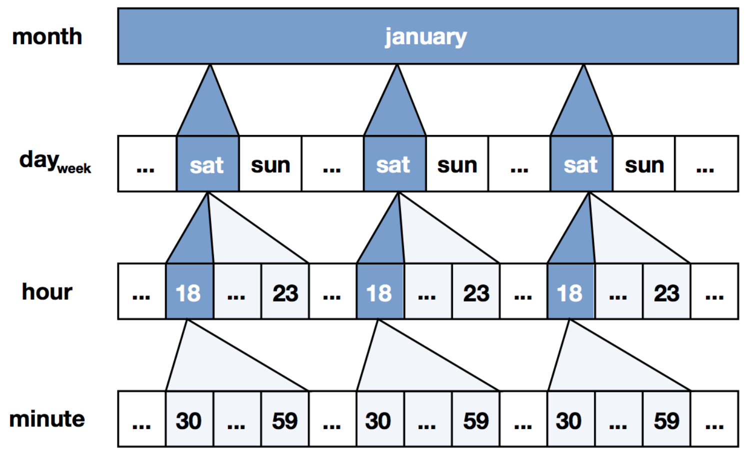


Figure 2.5: Drilldown performed (w.r.t. one of the months) when a query groups-by month over all Saturdays from 18:30 to 23:59.

query is presented in Figure 2.4. The query is executed by first drilling down starting from the 0-dimensional cuboid of the data cube. At each successive resolution r, the constraint values for that particular resolution (Line 4) are evaluated. Given a constraint in resolution r, the sub-arrays in B_r satisfying these constraints (within the given time range $[t_1, t_2)$) are first identified, and a drilldown is performed *only* with respect to these sub-arrays (Lines 10–12). Intuitively, a *drilldown* corresponds to expanding the cuboid by increasing its dimension by one. Figure 2.5 illustrates this procedure on a query that requires a group-by on *month* over all Saturday nights (18:30hrs–23:59hrs). For the *hour 18*, the execution drills down up to the *minute* resolution, and for the other hours in the constraints, only upto the *hour* resolution.

The drilldown is recursively repeated until the constraint in C at the finest resolution, r_c, is satisfied. Let $r_g \in G$ be the finest resolution on which a group-by is performed. If $r_g \prec r_c$, then a drilldown is further performed until r_g. On the other hand, if $r_c \prec r_g$, a rollup is performed until r_g. Intuitively, a *rollup* decreases the dimensionality of a cuboid by one by aggregating over one of its dimensions. At this stage, the group-by is performed recursively over all resolutions in G starting from r_g and rolling-up to coarser resolutions. At each resolution, the elements of the filtered (and previously grouped-by) sub-array are aggregated into the query result (Line 14).

Range Query. Time Lattice also supports range queries over time series data. A range query is used to query for the time series within a given time interval at an optional user specified resolution. This query is primarily intended for the visual exploration of the time series. A resolution coarser than the original resolution of the time series returns the computed aggregates. It is common for the visualization system to control the resolution specified in the query depending on the available screen space and the time constraint. For example, when visualizing a large time series, the screen space restricts each pixel to cover a time interval larger than a single unit of time. So, the system might choose to visualize the maximum value within the time interval corresponding to each pixel in order to obtain a big picture of the time series (analogous to the level of detail rendering used for terrains, which shows only larger mountains when the camera is distant, and increases detail as the camera moves closer to the scene).

The result for a query having time constraint $[t_1, t_2]$ and resolution r is simply the sub-array of B_r from $\alpha_r(t_1)$ to $\alpha_r(t_2)$.

2.2.3 Extensions

Handling Discontinuous Time Series. We have so far assumed that the given time series is continuous and without gaps. This, however, need not be true in practice. For example, a sound sensor could malfunction, and hence stop transmitting data. This would result in no data from the sensor until it is corrected. Such a situation can be handled in two ways. One could simply "fill" the gaps with a default value denoting lack of data. In this case, when aggregations are performed at a coarser resolution, these values should be dealt with appropriately. The other option is to maintain separate Time Lattices for each contiguous time series. In this case, the querying approach would be modified to perform queries over all Time Lattices whose time interval intersects the query time range, and combine the multiple results into a single result.

In our current implementation, we chose the former since the time interval between failure and replacement of sensors is typically small, resulting in a small memory overhead due to the "filling" operation. However, for cases where this gap can be significant, we advise the use of multiple Time Lattices.

Supporting Multiple Aggregations. Time Lattice supports the use of any *aggregable* measure. Here, a measure is said to be aggregable if its sufficient statistics can be expressed as a function of commutative and associative operators [73]. Thus, it allows an aggregation at a coarser resolution to be computed purely using the immediate finer resolution (and hence not using the raw data at all). In addition, measures such as median or percentiles can also be approximated by maintaining a histogram associated with each bin. The size of this histogram can be adjusted depending on the available memory and accuracy requirements.

The low memory requirement of Time Lattice further allows the addition of more advance summaries, as long as they are aggregable. For instance, each bin can have a *tdigest* [74] associated with it, so that holistic measurements such as quantiles can also be computed within an error threshold. As shown later in Section 2.5, it is also easy to add domain specific measures to the data structure.

Supporting Joins. Oftentimes, the analysis of a time series might require a join with another time series. For example, when analyzing the decibel level time series from a sound sensor, the domain expert might want to consider only time periods when there was significant rainfall (precipitation greater than a given threshold). Here the rainfall data would be represented by a second time series, say f'. To support such a join, we additionally store a histogram corresponding to f' in each element of a cuboid as follows. The bins of this histogram correspond to the range of f'. Consider one such histogram bin having range $[f'_1, f'_2)$. The value stored in this bin is equal to the aggregate of $f(t)$ where $t | f'_1 \leq f'(t) \leq f'_2$. Figure 2.6 illustrates once such histogram for the above rainfall example.

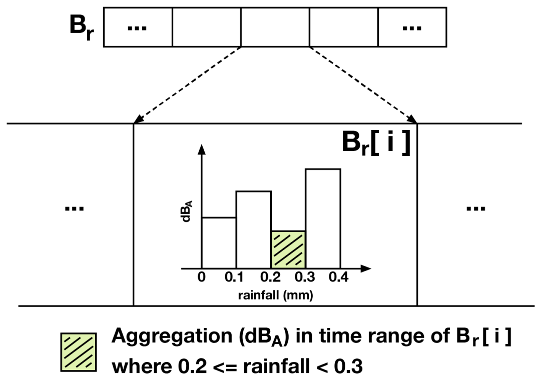

B_r
...
B_r[i]
dB_A
0
0.1
0.2
0.3
0.4
rainfall (mm)
▨ Aggregation (dB_A) in time range of B_r[i] where 0.2 <= rainfall < 0.3

Figure 2.6: An additional histogram corresponding to a second time series is stored in the cuboids to support joins in queries.

Note that the resolution of f' does not need to be the same as that of the time series of interest f. If the resolution of f' is finer than that of f, then f' is appropriately aggregated. If instead, it is coarser, then f' can be extrapolated to support constraints in a finer resolution. In our current implementation, we assume that the join condition based on f' is not coarser than the group-by resolution of the query. The case when the condition is coarser than the group-by resolution can be supported by storing the aggregate measure corresponding to f' as well in the Time Lattice, and drilldown performed only when an array element satisfies the condition.

Extended Materialization. Depending on the size of the underlying time series, queries could still be expensive depending on the query constraints. In such cases, selectively materializing more nodes can greatly help speed up the query execution. If frequently posed queries involve a group-by different from the ones materialized, then that corresponding cuboid is materialized.

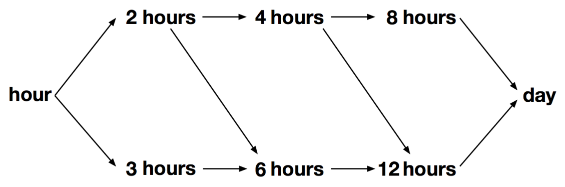


Figure 2.7: Expanded materialization between the resolutions *hour* and *day*. Recall that the array corresponding to *day* is shared for the resolutions day_{week} and day_{month}.

On the other hand, if frequently posed queries involve similar constraints along a single resolution, then it might be more beneficial to add a new dimension, and materialize that resolution accordingly. For example, users might frequently pose queries with constraints in hour of day to study patterns during different times of the day such as during peak hours in the morning and / or evening. One such query would be to obtain the aggregated behavior during peak morning hours (say 8 a.m. to 11 a.m.) grouped by the days of the week. To execute the queries, several sub-arrays are processed after filtering B_{hour}. As the size of the time series keeps increasing, this overhead could become significant. In such cases, a new coarser resolution can be introduced.

For a resolution r, there can be $a_r - 2$ possible resolutions that can be added corresponding to the possible time ranges. In the above example, this resolution would lie between *hour* and *day* with respect to the partial order $\prec$. There are $a_{day} - 2 = 22$ such possible resolutions ranging from 2 hours to 23 hours. If all of these resolutions are materialized, then it increases the size of the Time Lattice by a linear number of cuboids.

Materializing all such nodes for all resolutions might not be necessary for the required analysis. Instead, we allow users to specify common queries, and choose the new resolutions to be materialized. By default, one could materialize resolutions corresponding to time intervals that are factors of a_r. Figure 2.7 shows one such materialization between the hour and day resolutions, where the time intervals of sizes 2, 3, 4, 6, 8, and 12 hours are materialized. Queries with constraints having a different interval size are then computed by using a combination of these resolutions.

2.3 Noise Profiler

Working with researchers from the SONYC project, we developed a prototype web-based visualization tool, Noise Profiler, that uses Time Lattice to help in the visual analysis of the SPL data obtained from the different sensors deployed in NYC. In this section, we first describe the SPL data followed by discussing the design of the Noise Profiler interface. We finally describe how Time Lattice was used to support the different features of Noise Profiler.

2.3.1 Sound Measurement Data

For the remainder of this chapter, we use sound pressure level decibel (SPL dB$_A$) data obtained from the different acoustic sensors. Here, the A denotes a frequency weighting that approximates the response of the human auditory system. This data is sampled continuously at *1 second intervals* from each sensor. As mentioned in Section 2, the sensor network used for this work consists of 48 deployed nodes spread across NYC. Thus, each sensor generates a time series having approximately 31.5 million points per year. As part of the analysis, the researchers are also interested in computing a metric called *equivalent continuous A-weighted sound pressure level* (L$_A$*eq*). L$_A$*eq* is the sound pressure level in decibels equivalent to the total A-weighted sound energy measured over a given time period. This metric is used when exploring / analyzing acoustic data over coarser time resolutions.

2.3.2 Desiderata

The two main tasks that the researchers in the SONYC project are interested in are: 1) specify, execute, and visualize OLAP analytical queries over the SPL data from across the city; and 2) compare live data with the summaries obtained from these queries. To accomplish this, we develop a web-based prototype system built on top of Time Lattice to satisfy the following requirements: 1) visually specify queries— this includes the ability to select the time period of the data to analyze, apply constraints over different time resolutions, and specify dimensions on which to perform group-by's; 2) ability to select and compare data from one or more sensors based on the location; 3) support for the $\mathrm{L}_{A}eq$ metric as the aggregate in the queries; and 4) visualize live data together with the results of the queries We now briefly detail the interface followed by describing the backend query processing that is handled separately by a server.

2.3.3 Visual Interface

The Noise Profiler interface consists of two main components—a query panel and a time series widget (see Figure 2.1).

Query Panel. The query panel (Figure 2.1 (right)) allows the user to visually frame the different analysis queries, and choose the measure of interest to be visualized. While framing a query, the user can set constraints at various resolutions (e.g., analysis over weekends would require a constraint on the day_{week} resolution, and night time would require a constraint on the hour of day resolution). Users can also specify the group-by dimension. The time range of interest for a query is specified by brushing on the summary view from the time series widget (described next). The query panel also has a *map widget* (Figure 2.1 (left)) that displays the location of the deployed acoustic sensors from which the user can choose the sensors of interest. In addition, the user can also choose between analysis mode and streaming mode. The analysis mode is primarily used for analysis of historical data, while the latter allows users to visualize streaming data together with analysis queries.

Time Series Widget. The user can create one or more time series widgets, called *time series cards*. Each card is composed of a summary view providing an overview of the entire time series, and a detailed view, visualizing the result from a query.

Users can select the time range of interest by brushing over the summary view. When no constraints / group-by's are specified, the query simply corresponds to a range query, and is visualized in the detailed view. We support the level-of-detail rendering by default (see Section 2.2.2). The resolution at which it is visualized is determined by the screen space (number of horizontal pixels) available. Thus, by zooming in (selecting a smaller time range) the users can see more details of the time series. When group-by's are present, the result of this query over the selected time range is visualized in the detailed view.

Users can select several sensors to be visualized on a single card, and the chosen query is executed on all time series corresponding to these sensors. The color of a time series indicates the sensor source on the map. When there are multiple time series cards, queries are specified separately for each of them, thus allowing the user to use multiple cards for comparing different scenarios (e.g., day time vs night time, or different clusters of sensors).

When working in streaming mode, the live data from the selected sensors is shown together with the plots resulting from the specified query. Here, the live data is visualized using a lighter hue of the sensor color (see Figure 2.1).

2.3.4 Query Backend

We implement a server-based backend so as to allow users easy access to the Noise Profiler through a web browser. For each deployed sensor, we maintain one Time Lattice data structure. Given a query and collection of sensors (that are selected by the user), the query is executed once for each of the sensors. Due to the low latency of the Time Lattice data structure, it is possible to perform such analysis interactively. Note that this would not have been possible using existing techniques given their performance. The information about each of the sensor (e.g., location, deployed time) is stored separately, together with a reference to the Time Lattice corresponding to it. Missing and / or invalid data (e.g., when a sensor goes down) is filled with a default value.

When creating the data structure, in addition to the default minimum, maximum, sum, and average measures, we also store information to compute the $\mathrm{L}_{A}eq$ metric that was required for the analysis tasks. We maintain one background thread per sensor which listens for new data and updates the Time Lattice accordingly.

2.4 Experimental Evaluation

In this section, we discuss results from our experiments evaluating the efficiency of the Time Lattice data structure.

2.4.1 Experimental Setup

Hardware Configuration. All experiments were performed on a workstation with a Intel Xeon E5-2650 CPU clocked at 2.00 GHz and 64 GB RAM.

Data Sets. We generate synthetic time series data sets for our evaluation. The time series is itself at the *second* resolution, and for each second, a random number from a uniform distribution is used as the value for each time step (second). We generate time series of different sizes depending on the experiment that is performed.

Table 2.3: Queries used in the experiments.

Q1	**select** *time series* **between** December 14, 1970 5:20 and February 3, 1972 9:20 **aggregated by** *hour*
Q2	**select** *time series* **group by** *hour*
Q3	**select** *time series* **where** time **between** 09:30 and 17:30 **group by** *day*
Q4	**select** *time series* **where** time **between** 09:30 and 17:30, month **in** [January, February, March] **group by** *hour, minute*

Queries. The four queries used in our evaluation are shown in Table 2.3. We chose these queries to cover the different scenarios that arise during the visual analysis of time series data. Query Q1 is a range query typically used in the exploration of time series, and queries for data within the given range to be visualized as an hourly time series. Q2 is a group-by query used to visualize the hourly patterns in the data. Q3 queries for the day time patterns in the data for every day of the week. The complexity of the group-by query in this case is increased by adding a constraint on time (i.e., day time range). The above two queries are typically used to study ambient noise patterns (see Section 2.5.1). Finally, Q4 further increases the complexity of Q2 and Q3 by adding an additional constraint, as well as another group-by dimension. This query provides detailed minute-wise day time patterns over winter months.

State-of-the-art Approaches. For a comparison of Time Lattice with the state of the art in Section 2.4.3, we use a combination of both data cube-based techniques as well as libraries and databases that are catered for time series data analysis.

In particular, for the data cube-based baseline, we use nanocubes [47] which is also available as open-source software. We did not choose hashedcubes [48] since the available implementation supports only "count" queries and cannot perform aggregation over attributes. Also, nanocubes has better query performance than hashedcubes [48], and hence provides a better baseline. To be fair, we only chose resolutions that are used by the test queries as dimensions while constructing the nanocubes data structure. This also allows the data structure to be more memory efficient. The resolutions included were: *year*, *month*, day_{week}, *hour*, and *minute*, and *hour-minute*. The last category gives the minute of the day having a value between 0 and 1440. This was required to efficiently support queries Q3 and Q4 that have constraints on the time of day.

With respect to time series databases, we chose those that support OLAP queries: PostgreSQL with the timescale [52] extension, InfluxDB [45] and KairosDB [46]. We created a hypertable and an index on the time dimension when using the timescale extension in PostgreSQL. For both InfluxDB and KairosDB, we created *tag columns* corresponding to the time dimensions used for querying (same as the ones used for nanocubes). In addition to the above, we also compare our data structure with the in-memory python library Pandas [75] that is commonly used by data scientists in the analysis of time series data. To enable efficient querying, we created a DataFrame with an index on the time dimension.

Software Configuration. The Time Lattice data structure was implemented using C++. For all the experiments, the Hasse diagram in Figure 2.3 was used to create the Time Lattice on the input data. Queries were executed 5 times, and the median timings are reported.

2.4.2 Scalability

We first study the scalability of Time Lattice with increasing data sizes with respect to both query evaluation time as well as data structure update time.

Data Structure Size. Figure 2.8 shows the size of the Time Lattice data structure for different time series sizes. Note that the size of the structure includes that of

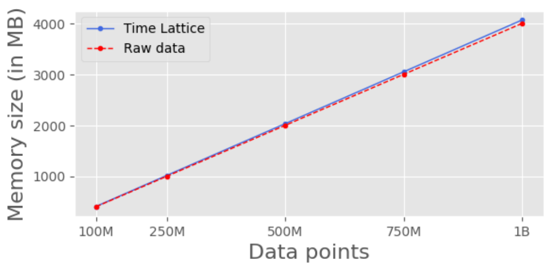


Figure 2.8: Size of Time Lattice within increasing time series size. Note that the additional memory overhead used for the data structure is **considerably** smaller than the data itself ($< 2\%$).

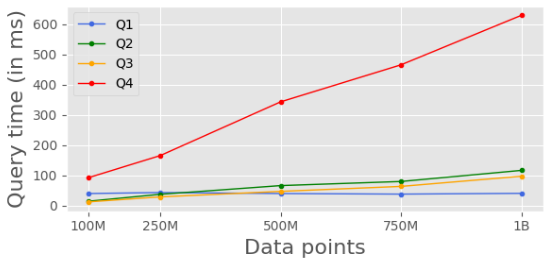


Figure 2.9: Query execution time for the four test queries as the size of the data increases.

the raw data, and the upper bound of additional memory overhead for the data structure is linear in the size of the data itself (see Section 2.2.1). In practice, as illustrated in the figure, this memory overhead is just a small fraction ($\approx 1.6\%$) of the underlying raw data.

Query Evaluation. Figure 2.9 shows the query evaluation time for the 4 test queries with increasing data sizes. Note that, except for Q1, the rest of queries cover the entire time series. As expected, one can see a linear scaling with data size. This is primarily due to the data structure size and query time trade-off in the design on Time Lattice. Since there is no cuboid materialized with respect to the group-by dimensions used in the queries, the query execution drills down to the finest resolution required, and the processing time is linear in the size of this dimension.

Q4, in particular, is an example of a pathological case query for our data structure due to the following reasons: 1) the time range selected does not align with the dimensions used thus requiring a drill down to a finer resolution during

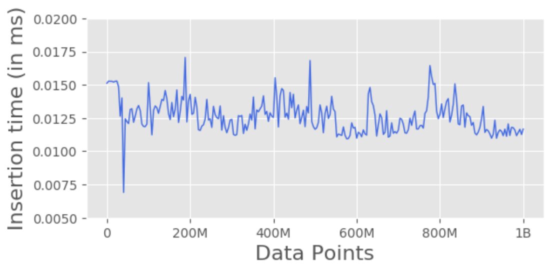


Figure 2.10: Average time per update. Note that the update time remains consistent (≈ 0.012 ms) even when adding new data to a Time Lattice built on a time series of size close to a billion points.

query evaluation (as a rule of thumb, query evaluation requiring only coarser resolutions are faster than those requiring finer resolutions); and 2) the *group-by* is on two dimensions–the corresponding cuboid is not precomputed. Thus, this aggregation has to be evaluated on the fly. Note that even for such complex group-by with multiple constraints over an entire time series having as large as *one billion* time steps, the queries take *less than 650 ms.*

Performance of Updates. Figure 2.10 shows the time to update the data structure with streaming data. For this experiment, we start with an empty Time Lattice, and insert data one time step at a time. The plot shows the average insertion time for an update with incoming data, and thus increasing data structure size as well. As can be seen from the figure, the time to update the data structure with new data is roughly constant, and around *0.012 ms.* This ensures that even if new data arrives at a frequency of every *millisecond*, our data structure will be *updated without any lag.*

Table 2.4: Comparing query response times of Time Lattice with existing approaches on a time series with 100M points.

	Size		Q1		Q2		Q3		Q4	
	(MB)	Increase	Time (ms)	Speedup	Time (ms)	Speedup	Time (ms)	Speedup	Time (ms)	Speedup
Time Lattice	397	—	40.5	—	15.0	—	12.8	—	92.4	—
Nanocube	41799	105X	116.0	2.9X	4.6	0.3X	2491.8	194X	40083.9	433X
Pandas	1600	4X	1670.1	41.2X	9355.1	623.6X	10399.3	812X	11070.6	119X
InfluxDB	412	1.03X	10574.6	261X	42913.5	2860X	35259.5	2754X	29058.0	314X
TimescaleDB	7867	19X	20385.1	503X	60206.4	4013X	130594.5	10202X	101036.1	1093X
KairosDB	1301	3X	229110.9	5657X	629886.4	41992X	240168.2	18763X	75267.1	814X

2.4.3 Comparison with State of the Art

Table 2.4 compares the performance of Time Lattice with the current state-of-the-art solutions. A time series of size 100 million seconds was used for this experiment. For all the approaches, except for Time Lattice, we had to add additional columns to the data corresponding to the resolutions before loading it. Time Lattice has the lowest space requirement, while nanocubes consumes the most memory. InfluxDB, which compresses the data comes a close second.

The table also shows the query execution times for the four test queries. The performance of Pandas and the three databases is significantly slower than that of Time Lattice. While nanocubes has good performance for Q1, it has the best performance for Q2. This is because Q2 is a straightforward group-by without any constraint or filtering on time. Since nanocubes is essentially a memory optimized data cube, this query is simply a lookup from the corresponding bin. On the other hand, when more complex constraints are imposed, the performance significantly degrades. To improve performance of nanocubes for the constraints involving time of day, one could additionally add a dimension to the data corresponding to it. However, when we tried to create the nanocubes structure with this additional dimension, it ran out of the available 64 GB memory.

2.5 Case Studies

In this section, we illustrate how Noise Profiler can be used in the visual analysis of multiple large time series data with a focus on understanding the acoustic noise patterns in NYC. In particular, we discuss three case studies performed by the researchers in the SONYC project.

2.5.1 Exploring Noise Patterns via Grouping

To better understand the acoustic conditions of the urban environment, long-term monitoring is required to capture the variations in SPL over different periods: minutes, hours, days, weeks, months and seasons. For example, noise enforcement agencies in cities typically assess a breach of the noise code by a given rise in SPL above the ambient background SPL. In cites, this ambient background SPL varies at many different temporal resolutions, thus it is important to understand these trends in order to better enforce local noise codes.

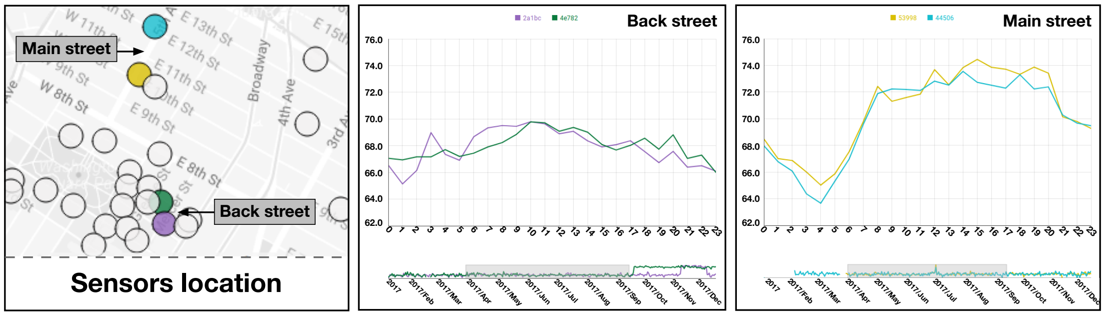


Figure 2.11: Comparing daily patterns of noise on back streets with noise on main streets.

Case Study 1: Location-wise Noise Patterns. In this case study, we were interested in exploring the data for global trends, in particular how the noise pattern changes throughout the course of a single day at different sensor locations. This question essentially corresponds to the following group-by query on the different sensors.

$$\textbf{\textit{select}}\ \textit{time series}\ \textbf{\textit{between}}\ t_1\ \textit{and}\ t_2\ \textbf{\textit{groupby}}\ \textit{hour}$$

Here, $[t_1, t_2)$ corresponding to the time period of interest. To do this, we first select sensors of interest into the same time series card and configure the above query in the query panel.

Figure 2.11 shows the results from 2 sensors on main traffic thoroughfares and 2 on quieter back streets. It thus required executing 4 group-by queries, each of which took 100 ms to execute. The morning rush-hour ramp up in dB_A level begins at the same time for each group of sensor locations, however, the main-street locations maintain a raised dB_A level until around 7 p.m., when the evening rush hour begins to trail off. The reduction in dB_A level after 1 p.m. for the back street sensors could suggest that these streets are typically less used for evening rush hour travel. The difference in dB_A level between the early morning (12 a.m.–5 a.m.) and peak daytime dB_A levels from 8 a.m.–7 p.m. is far more pronounced at ≈7dB for the main-street locations compared to ≈2dB for the back-street locations. This highlights the impact of traffic noise on the main-street locations, when compared with back-street locations.

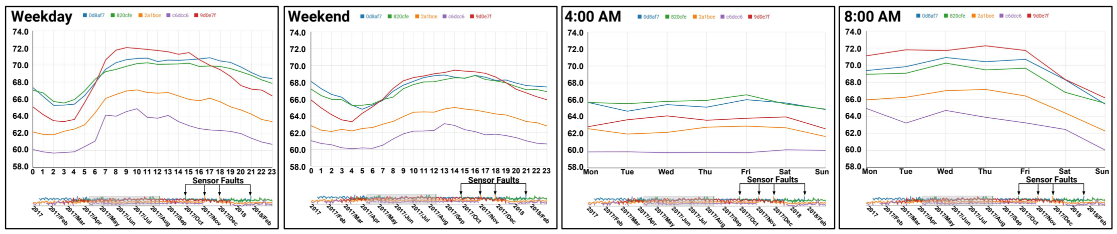


Figure 2.12: The two plots on the left show noise patterns on weekdays vs. weekends on diverse locations around Washington Square Park and Central Park. The two plots on the right show the weekly noise patterns for 4 a.m. and 8 a.m.

Case Study 2: Weekday vs Weekend Patterns. On weekends the daily dB_A levels throughout the day would intuitively exhibit a different trend to those on weekdays. Knowing these differences allow city agencies to better understand the evolution of ambient background levels at different periods of the day and week. Figure 2.12 shows separately the weekday and weekend daily dB_A level evolutions, aggregated by hour for 5 sensors across varying locations. This shows an $\approx$1dB difference between weekday and weekend peak dB_A levels, highlighting the raised weekday levels. Of note is the increased gradient on the ramp-up period from early morning to peak rush hour on the weekday plot compared to that of the weekend plot. That is, during weekdays, the noise levels increase sharply between 4 a.m. and 7 a.m. On the other hand, during the weekend, the noise levels start increasing later, at 5 a.m., and take until 2 p.m. to reach peak levels. A key point that is apparent from these plots is the $\approx$ 1 hour later shift in this ramp up at weekends suggesting that noise making activities begin later and take longer to increase over time.

By visualizing these hourly noise patterns, the above analysis provides the hours of interest to investigate more closely. In particular, while the ramp-up patterns are clear, it is still not straightforward to make out how these hours vary over the different days of the week. This can be visualized using the following query template:

select *time series* ***between*** t_1 *and* t_2 ***where*** *hour=4am*
groupby day_{week}

Figure 2.12 also shows the weekly noise patterns at 4 a.m. and 8 a.m. respectively, allowing us to explore a different perspective of this data. Note how the noise level

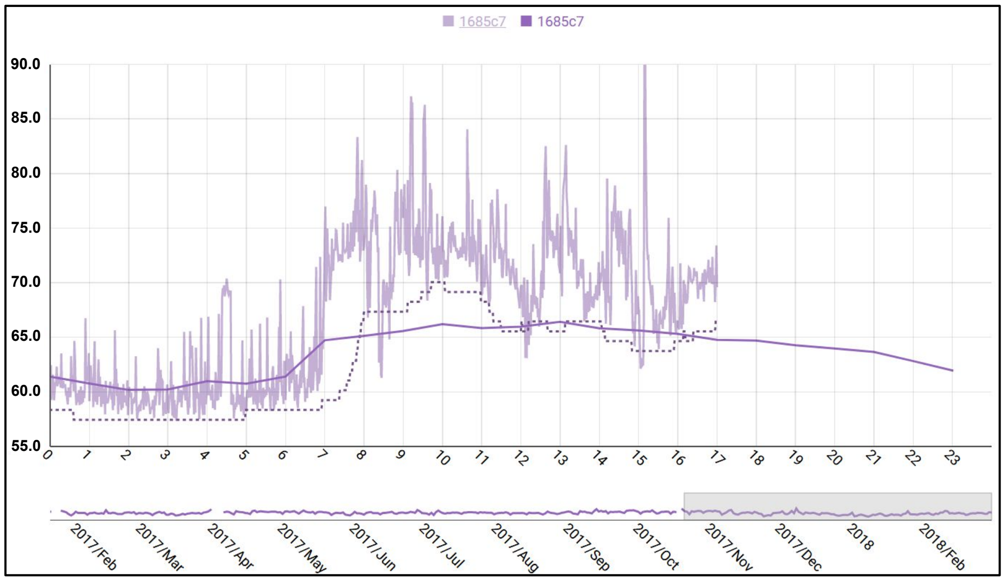


Figure 2.13: Comparing live data with two different ambient noise baselines for a given sensor.

at 8 a.m. is relatively constant on weekdays, but is lower on Sundays as compared to Saturdays. On the other hand, it remains consistent throughout the week at 4 a.m. Each of the queries posed to obtain the above visualizations took on average 80 ms per sensor.

These findings can provide valuable information to city agencies looking to understand the temporal characteristics of dB_A levels at different days of the week. For example, construction permits are generally not issued for work over weekends to reduce the impact on city inhabitants. However, special out–of–hours permits can be requested for weekend work. With knowledge on the temporal evolution of dB_A levels on weekends for a particular location, these permits can be time limited to periods of high ambient dB_A levels, reducing the impact of construction noise on local residents.

Finally, an unexpected outcome of the visual analysis process described above was the identification of erroneous sensor data due to sensor faults as seen in the excessive and continuous raised dB_A levels that can be seen in the summary view in Figure 2.12. The visual interface allowed us to quickly and easily exclude this erroneous data from the analysis. This kind of sensor data anomaly identification is crucial when maintaining a sensor network of this scale.

Case Study 3: Ambient Noise Baselines. In NYC, the indication of a noise code violation is given when a noise source exceeds the ambient dB_A level by 10dB. This prompts city agency inspectors to investigate further into the offending noise source to determine the extent of its breach of the noise code. This ambient level measurement is typically carried out using an instantaneous “eye–ball” measurement using a sound level meter while the offending noise source is not operating. Agency inspectors can also request that these noise sources be switched off to gain a more representative ambient measurement. The issues here are: (1) that this instantaneous ambient measurement may not be that representative of the area, as experienced by its inhabitants / noise complainants over extended periods of time on that day of the week, and (2) that a passive acoustic monitoring network does not have the ability to request the temporary shutdown of noise sources.

Given (1), it is important to consider an ambient background level computed over a more representative period of time, in order to decrease the impact of short lived noise sources and day of week influences. This is naturally captured by a group-by query. Figure 2.13 shows such a case, with the ambient level computed as the hourly average over the last 11 months, considering only weekdays. Note that the result of this query gets continuously updated with new incoming data. Prior to using the Noise Profiler interface with the Time Lattice data structure, we computed the ambient noise as the 90th percentile over a much shorter and “temporally naive” period of 2 hours, as in, it does not consider the holistic ambient level of this location, during the same period of time over multiple past instances of this period. This is illustrated using a dashed line in Figure 2.13. Note that that the ambient noise computed using this approach follows the same trend as the actual instantaneous dB level, resulting in a less representative ambient background level measurement. Thus, the use of select historical data for ambient level calculation, therefore addresses issue (2), providing a representative ambient background level measure for effective real–world noise code enforcement. Using Time Lattice, computing both the group-by queries as well as the 90th percentile measure took only 150ms even as the the data structure is simultaneously updated with incoming data.

Figure 2.1 presents another example showing the weekday hourly noise patterns of two different sensors. One sensor (blue) is located close to a main road (Broadway Av.), and presents a relatively constant dB_A level throughout the hours of the day. The other sensor (orange) is close to a major construction site, with a higher dB_A level during regular construction hours of 7 a.m. to 5 p.m. Also notice a temporary dip in the live noise level around lunch time.

As this use case demonstrates, the combination of OLAP queries over long time-periods and live streaming data can be used to better guide city agents when issuing noise code violations (e.g., construction sites operating outside of their allotted construction hours), as well as to better understand the noise profile of certain regions.

2.5.2 Feedback

As researchers using the Noise Profiler, we found several advantages in using the proposed system. The primary among them was the ability to seamlessly deal with high resolution SPL data covering large time periods. The high temporal resolution of the acoustic data streamed from our noise sensor network results in vast amounts of data. The frequently short-lived nature of urban noise events mean that all of this data needs to be considered when determining the effects of this noise on city inhabitants. We were typically limited to interacting with small subsets of the data, especially when dealing with a duration of more than a few days due to the limitations of our current tools (e.g., Pandas). The addition of the ability to interactively explore historical data simultaneously from multiple sensors helps tremendously, as now we can make more informed decisions based on the acoustic conditions at multiple locations As shown in the last case study, OLAP queries also allow the computation of a more meaningful baseline for ambient noise level measurements, a clear improvement over our previous “temporally naive” baseline. This in particular would be of great benefit to city agencies tasked with urban noise enforcement to better understand sources noise levels with respect to a representative ambient baseline. In addition to this, the Noise Profiler would allow a noise enforcement officer to query the periods at the very start and end of the allowed construction times of 7 a.m. and 6 p.m. Construction sites that begin early or end late can be scheduled a visit, optimizing agency resource allocation to the places that matter.

We also recently demonstrated the Noise Profiler prototype to experts from NYC's Department of Environmental Protection (DEP). While they were impressed with the analysis capabilities, especially the responsiveness in querying and handling data from multiple sensors, they found the general query interface a little overwhelming. In particular, they want to simplify the query interface by making it more focused on the typical queries that they repeatedly perform. We are currently in the process of making our system live for them to use.

2.6 Discussion

In this chapter we presented Time Lattice, a memory efficient data structure to efficiently handle complex OLAP queries over time series data. By selectively materializing a subset of the data cube based on the intrinsic hierarchy of the time resolutions, it allows for a linear memory overhead and also supports constant amortized time updates to the data structure. We also developed Noise Profiler, a web-based visualization framework that uses Time Lattice to allow the interactive analysis of data captured from acoustic sensors deployed around New York City.

While our current implementation can easily handle time series having a billion points interactively, as the data size keeps increasing, interactivity might not always be possible. However, many steps in the query execution process can be parallelized. In future, we intend to explore both CPU as well as GPU-based parallelization strategies, which can enable sub-second response times even with time series having several billions of points.

Chapter 3

Spatiotemporal Data: Real-Time Ranking of Geotagged Keywords

Ranks and lists play a major role in human society. It is natural for us to rank everything, from movies to appliances to sports teams to countries' GDPs. It helps us understand a world that is increasingly more complex by only focusing on a subset of objects. There is probably no better way to describe a decade than by ranking its most popular songs or movies. One just needs to look at the *Billboard Hot 100* of any year to gain insight into how the majority of society used to think and behave. *High Fidelity* [76] describes a person obsessed with compiling top-five lists for every occasion; as summarized in the book, the act of ranking gives structure to life, by clarifying the past [77].

With the ever-increasing amount of user-generated content found online, ranks have never been so popular to our cultural landscape. *"What's trending"* has become a commonplace phrase used to capture the spirit of a time by looking at the most popular Twitter hashtags. The same way that the most popular songs can be used to describe the *zeitgeist* of a year or a decade, *what's trending* can be used to describe the spirit of a day or even an hour.

In addition to the deluge of new user-generated data, the ubiquity of GPS-enabled devices provides further insight into the people creating this content by providing, in many cases, their location when posting a message to Facebook, uploading a picture to Flickr, checking into Foursquare at their favorite restaurant, or posting a review about a new product they just bought. The location information provides a much more interesting, and more complicated, version of the ranking problem. Now, not only are we interested in what is trending over time, but also over space, which can range from the entire world all the way down to a city street. What might be trending in one neighborhood of a city, may be completely different from other neighborhoods in other cities, and these may be completely different from the overall global trend across the country. Thus, the ability to explore these ranks at different spatial and temporal resolutions is important for understanding our world and the people in it.

A system that can efficiently process large amounts of data and come up with answers in a few minutes or seconds can be applied to many important problems, including those described here. In recent years, though, with the explosion of data gathering capabilities, there is a growing demand for interactive tools with sub-second latencies. Problems for which no automatic procedure exists that can replace human investigation of multiple (visual) data patterns require exploratory tools that are driven by a low latency query solving engine. Making an analyst wait too long for the query answer might mean breaking her flow of thought, and be the difference between capturing an idea or not.

We have recently seen a growth in research revisiting previous techniques and proposing new variations for solving exactly the problem of low latency queries for the purposes of driving visual interactive interfaces. From the perspective of enabling fast scanning of the data at query time, works like MapD [78] and imMens [79] use the parallel processing power of GPUs to answer queries at rates compatible with interactive interfaces. From a complementary perspective, Nanocubes [47] describes how to pre-aggregate data in an efficient but memory intensive fashion to allow for light-weight processing of queries in real-time.

In this chapter, we also follow this path of describing techniques for low-latency querying of large data for exploratory visualization purposes. We propose an extension to the efficient pre-aggregation scheme described in Nanocubes [47] to cover an important use case that was not discussed: interactively ranking top objects from a large data collection incident to an arbitrary multi-dimensional selection. For example, we would like to answer queries such as: "What are the top Flickr tags?" for manually selected spatial regions such as Europe, California, Chicago, or Times Square. Furthermore, we want to be able to determine how the popularity of these tags evolves over time, as dictated by the end-user's interests, all at interactive rates.

We show that the state of the art is not able to compute such queries fast enough to allow for interactive exploration. TopKube however, is a rank-aware data structure that is able to compute top-k queries with sufficiently low latency to allow for interactive exploration. Through a set of benchmarks which we make available publicly, we show that our proposal is up to an order of magnitude faster than the previous state of the art. More specifically, we can summarize our contributions as the following:

- A rank-aware data structure that allows for interactive visual exploration of top-k queries, with results up to one order of magnitude faster than previous work.
- A set of case studies that demonstrate the utility of our method using real-world, publicly available datasets, ranging in size up to hundreds of millions of records.
- A new set of publicly available benchmarks for others to validate their methods and compare to our own.

3.1 Related Work

The challenge of visualizing large datasets has been extensively studied over the years. Most techniques usually propose some form of data reduction: they try to aggregate a large number of points into as few points as possible, and then visualize that smaller aggregation. The original dataset is reduced to a smaller, sometimes bounded, version. Such reductions try to convey most, if not all, of the properties of the original dataset while still being suitable for interactive visualization. Perceptually approximate techniques [80, 81], which utilize data reduction, maintain interactivity while returning visual approximations that approach the exact results. Sampling [82], filtering [83] and binned aggregation [84] are among the most popular reduction techniques. Even though sampling and filtering reduce the number of items, it comes at the price of missing certain aspects of the data, including outliers. As pointed out by Rousseeuw and Leroy [85], data outliers are an important aspect of any data analysis tool. Binned aggregation, however, does not have such limitations. The spatial domain is divided into bins, and each data point is placed into one of those bins. As such, binning does not remove outliers and also preserves the underlying density behavior of the data.

The *visual* exploration of large datasets, however, adds another layer of complexity to the visualization problem. Now, one needs to query the dataset based on a set of user inputs, and provide a visual response as quickly as possible, in order not to impact the outcome of the visual exploration. In Liu and Heer [30], the authors present general evidence for the importance of low latency visualizations, citing that even a half second delay in response time can significantly impact observation, generalization, and hypothesis rates. Systems such as imMens [79], Nanocubes [47], and DICE [86] leveraged a data cube to reduce the latency between user input and visualization response. Data cubes have been explored in the database community for a long time [54], but in the visualization community, they were first introduced in 2002 by Stolte et al. [87, 88]. All of these techniques, however, are limited to simple data types, such as *counts*. They were designed to answer queries such as: *"How many pictures were uploaded from Paris during New Year's Eve?"* or *"How many GitHub commits happened on the West Coast?"*. Our data structure goes beyond that. We aim to answer more detailed queries, such as *"What were the most popular image tags for all the pictures uploaded from Paris?"* or *"What are the GitHub projects with most commits in the West Coast?"*.

The notions of ranking and top-k queries were also first introduced by the database community. Chen et al. [89] present a survey of the state of the art in spatial keyword querying. The schemes can be classified as spatial-first, text-first or a combination. Spatial-first structures create hierarchical spatial structures (e.g., grids [90], or R-trees [91]), and insert textual data into the leaf nodes. Text-first structures organize the data by considering the textual elements, and then linking a spatial data structure to it, keeping track of its occurrence in the space [90, 92]. Combined structures create data structures that handle both spatial and textual elements simultaneously [93]. The previous data structures, however, focus on building indexing schemes suitable for queries where the universe of keywords is restricted. In other words, given a set of keywords, rank them according to their popularity in a region. If there is no keyword restriction or the number of restricted keywords is too large, then such proposals become unfeasible. Our proposal is much broader: we are able to compute the rank of most popular keywords even if there is no keyword restriction.

Rank-aware data cubes were also proposed in the database community. Xin et al. [94] defined the *ranking cube*, a rank-aware data cube for the computation of top-k queries. Wu et al. [95] introduced the ARCube, also a rank-aware data cube, but one that supports partial materialization. Our proposal differs from them in two major ways: we specialize our data structure to better suit spatiotemporal datasets, and we demonstrate how our structure provides low latency, real-time visual exploration of large datasets.

Another related database research area is top-k query processing: given a set of lists, where each element is a tuple with key and value, compute the top-k aggregated values. Several memory access scenarios led to the creation of a number of algorithms [96]. The NRA (no random access) algorithm [97] considers that all lists are sorted and that the only possible memory access method is through sorted access (i.e., read from the top of each list). The TA (threshold algorithm) [97] considers random access to calculate the top-k. More recently, Shmueli-Scheuer [98] presented a budget-aware query processing algorithm that assumed the number of memory reads is limited. We propose a different top-k query processing algorithm, suitable for our low latency scenario. We show that, due to the high sparsity of the merged ranks, past proposals are not suitable.

Similar to what we are trying to accomplish, Birdvis [99] displays the top words in a given region; however, the technique does not scale to more than a few hundred words. Wood et al. [100] also present a visual exploration of tags in maps, using a standard MySQL database; but they are limited to less than 2 millions words and they do not present any time measurements.

Although orthogonal to the core investigation done here, we lastly mention some of the visualization techniques used strictly for display purposes of ranked objects. RankExplorer [101] proposes a modified theme river to better visualize ranking changes over time. Lineup [102] presents a visualization technique for multi-attribute rankings, based on stacked bars. A Table [103] proposes an enhanced soccer ranking table, with interactions that enable the exploration of the data along the time dimension. Our work enables these types of visualizations to be driven at interactive rates, rather than competes with them by offering a new vis method.

3.2 Motivation

We begin with a simple example. Assume a data analyst is studying shots in National Basketball Association (NBA) games from a table similar to Table 3.1 (only three rows shown). Every row represents a shot in a game. The first row indicates that LeBron James from Cleveland took a shot in the 5th minute of a game from the court coordinates $x = 13$, and $y = 28$; the shot missed the basket and he scored 0 points with that shot. The second row represents a successful two point shot by Rajon Rondo from Boston, and the third row represents a successful three point shot by LeBron James.

Table 3.1: Example of shot statistics in NBA games.

team	player	time	pts	x	y
CLE	L. James	5	0	13	28
BOS	R. Rondo	5	2	38	26
CLE	L. James	7	3	42	35

3.2.1 Generality vs. Speed

With such data and a SQL-like interface, an analyst could generate many insightful aggregations for this data. The query:

select *player, count(∗) as shots* ***group by*** *player* ***order by*** *shots* ***desc limit*** *50*

would generate a ranked list of the 50 players that took the most shots, while the following query would rank the top 50 players by points made:

select *player, sum(pts) as spts* ***group by*** *player* ***order by*** *spts* ***desc limit*** *50*

Although an SQL engine provides a lot of power and flexibility, these characteristics come with a cost: solving certain queries may require scanning all records stored in a potentially large database. Such an expensive computation may result in an analyst wasting precious time, or worse, breaking a flow of ideas and losing a potential insight. To solve these expensive queries more quickly, there are two alternatives: (1) expand the raw computational power of the query engine (e.g., using more CPU or GPU cores); or (2) anticipate important use cases and precompute information from the data so that query solutions can be composed more cheaply during data exploration. The idea of a *data cube* is essentially that of (2): make an explicit encoding of multiple aggregations a user might query later (e.g., store all possible group-by queries on a table using `SUM`). Note that we can think of solution (2) as being implemented by first running a big scanning query like in (1), but with a goal of storing enough information to allow for a fluid experience later.

In the Nanocubes paper [47], the authors follow this second approach, and observe that for spatiotemporal datasets with a few categorical dimensions, by carefully encoding the aggregations in a sparse, pointer-based data structure, they could represent, or *materialize*, entire data cubes efficiently. They report on multiple interesting use cases where these data cube materializations of relatively large datasets could fit into the main memory of commodity laptops, enabling aggregations (of the pre-defined measure of interest), at any scale and dimension (from city blocks to whole countries, from minutes to years), to be computed at interactive rates.

From a very high level, the Nanocubes approach is all about *binning* and *counting*. Each dimension in a Nanocube is associated with a pre-defined set of *bins*, and the technique consists essentially of pre-computing a mapping between every combination of bins (from the different dimensions) into some *measure* of all the records incident to that combination of bins. In the simple NBA example, the measure of interest could be the number of shots taken, but in other datasets this could be the number of lines of code, cell phone calls, hashtags, miles driven, or any other simple data value. Thus, the Nanocubes approach trades off flexibility (by requiring the measure of interest and binning scheme to be determined ahead of time) for interactivity (by providing rapid query responses during visual exploration of the data). It should also be noted that the original data records are not stored in the bins (just the counts) and therefore are not available as part of the Nanocube data structure.

The main drawback of in-memory data cubes as proposed by [47] is clear: even a minimal representation of a data cube tends to grow exponentially with the addition of new dimensions. In other words, there is no miracle solution to the curse of dimensionality. On the other hand, from a practical perspective, there seems to exist a sweet spot in terms of computing resource utilization where the

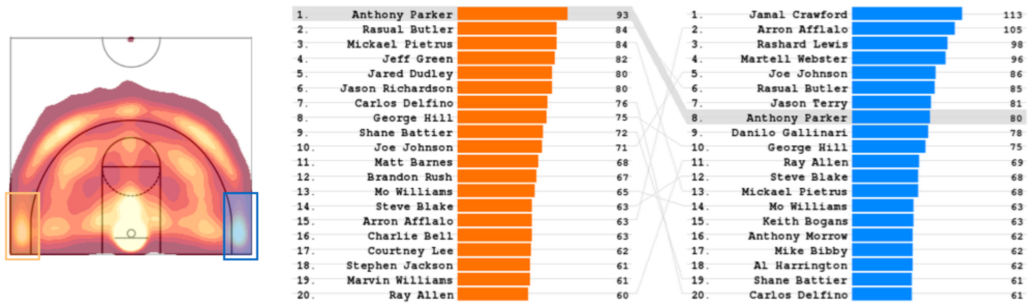


Figure 3.1: Ranking NBA players by number of shots from the left 3-point corner (orange) and right 3-point corner (blue) for the 2009-2010 season. The left image is a heatmap of all shots: brighter colors indicate more shots were taken from that location. The hotspot clearly identifies the basket.

application of in-memory data cubes can really be the driving engine of interactive exploratory experiences that would otherwise require prohibitively larger amounts of infrastructure.

With this context in mind, in this work we want to investigate the use of in-memory data cubes to drive an important use case for visualization that was not efficiently solved by Nanocubes. Namely, if we perform a multi-dimensional selection that contains millions of objects, how can we efficiently obtain a *list* of only the top-k most relevant objects with respect to our measure of interest. For example, consider the selections we made on the basketball court example in Figure 3.1. We want to compare the top-20 players that take shots on the left 3-point corner (orange) versus players that take shots from the right 3-point corner (blue). In this case, knowing that there are only a few hundred players in the NBA each year, it would not be computationally expensive to scan all players to figure out the top 20, but there are many other cases such as GitHub projects, Flickr images, and Microblog hashtags, where having to scan millions of objects can result in unacceptable latencies.

3.3 Multi-Dimensional Binning Model

In the following, we establish our own notations for well-known concepts in the database literature so that we can have a minimal, self-contained, and precise language to refer to when presenting the various data structures.

The core abstraction used by Nanocubes [47], and shared by our proposal of TopKube, is that of associating records with bins in multiple dimensions: a *multi-dimensional binning model*. For example, it is natural to associate the NBA shot records in Table 3.1 with dimensions `team`, `player`, `time`, `points`, and `location` (note we chose here to combine columns `x` and `y` into a single `location` dimension). Each player possibility can be associated with its own bin in the `player` dimension. Team and points are handled similarly. For time, we could choose a one minute resolution and have each minute of the game be a bin in the `time` dimension; for location, we could have a 1ft. $\times$ 1ft. grid model of the court area and have each cell in this grid be a bin in the `location` dimension. Note that, in this modeling, each record is associated to one and only one bin in each dimension. More abstractly, in our formalism, we assume each dimension i has a *set of finest bins*, denoted by B_i', and a record is always associated to a single finest bin in each dimension.

In addition to the set of finest bins B_i' associated with dimension i, we define the notion of *coarser* bins, or bins that "contain" multiple finer bins. For example, in the location dimension, we could group adjacent finest grid cells into 2x2 groups and make each of these groups a coarser bin cell in the `location` dimension. The interpretation of coarser bins is simply that if a record is associated with a finer bin b then it is also associated with a coarser bin that "contains" b. In the binning model we define here, we assume that the *set of all bins* B_i associated with dimension i forms a *hierarchy* $H_i = (B_i, \pi_i)$ where its leaves are a partition of finest bins B_i'. The containment function, $\pi_i : B_i \to B_i$ associates every bin b to another bin $\pi_i(b)$ which is either the smallest (i.e. finest) bin in H_i that contains b (in case $b \neq \pi_i(b)$) or it is the coarsest (or root) bin in H_i (in case $b = \pi_i(b)$).

An *n-dimensional binning schema* S is defined as an n-ordered list of hierarchies: $S = (H_1, \ldots, H_n)$. In order to extend the *finest sets of bins* for the `player` dimension, $B'_{\texttt{player}}$, into a valid bin hierarchy $H_{\texttt{player}} = (B_{\texttt{player}}, \pi_{\texttt{player}})$, we could include an additional coarse bin that serves as the root of this 1-level hierarchy by making all bins in $B'_{\texttt{player}}$ its direct children. This is indeed the natural way to model categorical dimensions with few classes. For dimensions where the number of finest bins is not small, it is best to use multi-level hierarchies so that data can later be accessed more efficiently in a level-of-detail fashion. For example, a natural way to model spatial dimensions is by using a quadtree bin-hierarchy; for a temporal dimension, in TopKube, we use a binary-tree bin-hierarchy. Given an n-dimensional binning schema S, we say that the Cartesian product $\mathcal{B} = B_1 \times \ldots \times B_n$ is the *product-bin set* of S, and that an element $\beta \in \mathcal{B}$ of this set is a *product-bin* of S.

To define a *multi-dimensional binning model*, it remains to formalize the notion of which records in a dataset are associated to which bins and product-bins. Let R be a set of records and S a binning schema. If we provide an *association function* $a_i : R \to B'_i$ for each dimension i that assigns a unique *finest bin* in B'_i for every record, we can naturally and uniquely define an association relation $A \subseteq R \times \mathcal{B}$ between records and product-bins. Here is how: (1) we say a general bin $b_i \in B_i$ is *associated with* record r if either $b_i = a_i(r)$ or b_i is an ancestor of $a_i(r)$ in H_i; (2) a product-bin $\beta = (b_1, \ldots, b_n)$ is associated with record r, denoted by $(r, \beta) \in A$ if b_i is associated with record r for $1 \leq i \leq n$.

A *multi-dimensional binning model* is thus a triple $M = (S, R, A)$, where S is a binning schema, R is dataset of records, and A is an association relation between records and product-bins $\mathcal{B}$ from schema S. Given a product-bin β we use $A(\beta)$ to denote its associated set of records in model M (i.e., $A(\beta) = \{r \in R : (r, \beta) \in A, \beta \in \mathcal{B}\}$). Analogously, we use $A(r)$ for a record r to denote its associated set of product bins (i.e., $A(r) = \{\beta \in \mathcal{B} : (r, \beta) \in A\}$).

3.3.1 Measure on a multi-dimensional binning model

The notion of product-bins in our model provides a way to refer to groups of records through their multi-dimensional characteristics. In our running NBA example, all shots of LeBron James would be specified by $A(\beta_{LJ})$, where the product-bin $\beta_{LJ} \in \mathcal{B}$ consists of the coarsest (root) bin in the bin-hierarchy of all dimensions, except on the player dimension where we would have the bin for LeBron James. If instead we want to refer to LeBron James' shots in the first minute of a game, we would replace the root bin in the `time` dimension of β_{LJ} with the bin for minute 1 of the game.

One natural approach to analyzing a set of records through their multi-dimensional characteristics is through some notion of "size". For example, instead of listing all NBA shots from the 3-pt left corner, we could simply be interested in how many shots happened in that region, or what the average or median distance from the basket was for all of those shots. A *measure* for a multi-dimensional binning model

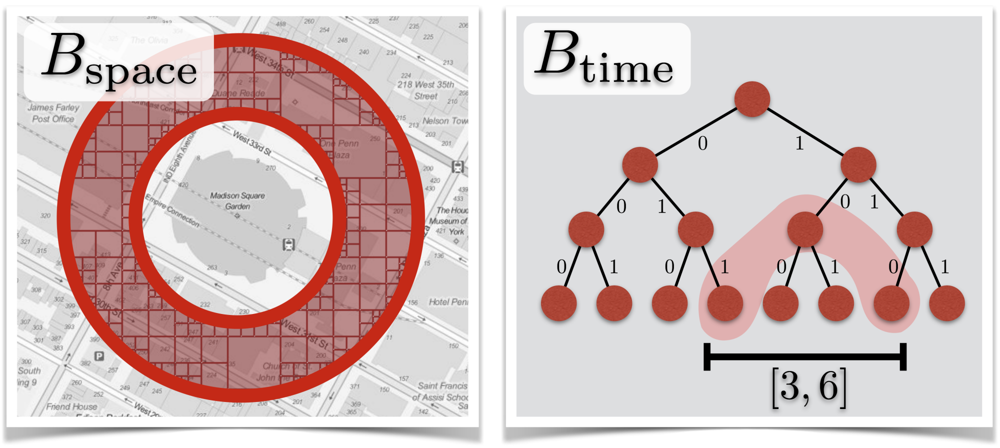


Figure 3.2: Dimensions of *space* and *time* are represented as bin hierarchies. (left) B_{space} is a quad-tree hierarchy: here we show a 624 bin selection around Madison Square Garden, NY; (right) B_{time} is a binary hierarchy; in red we show 3 bins corresponding to the interval $[3,6]$.

M is simply a real function $\mu : \mathcal{B} \to \mathbf{R}$ that associates a number to any product-bin β of M, which captures some notion of "size" for the set of incident records $A(\beta)$. In the target applications we are interested in, we want to access measure values not for just one product-bin at a time, but for sets of product-bins that are semantically meaningful together. For example we might be interested in the spatial region on the left of Figure 3.2 that consists of multiple bins. In general, one cannot derive the measure value of the union of a set of product-bins by combining the measure values of the individual product-bins. The median distance of an NBA shot is such an example: we cannot derive the median of the union of two sets of values by knowing the median of each individual set. We avoid this problem here by restricting our universe to those of *additive* measures only. We start with a real function $\mu : R \to \mathbf{R}$ that associates a number to each record from model M and extend this function to the whole set of product-bins by using additivity $\mu(\beta) = \sum_{r \in A(\beta)} \mu(r)$. Additive measures can naturally count occurrences (e.g. how many records) by making $\mu(r) = 1$, or measure weight sums by making $\mu(r) = w_r$. In addition to scalars, we can also generalize additive measure to produce real vectors. For example, by making $\mu(r) = (1, w_r, w_r^2)$ additivity will yield a 3d vector on any product-bin and union of product-bins (just sum the vectors). In this 3d measure example, it is possible to post-process the vector entries to derive mean and variance of weights for any set of product-bins (mean: divide second entry by first entry). Correlations can also be derived by post-processing an additive measure [73]. In the remainder of this chapter we assume simple additive scalar measures. We do not deal with post-processed ones.

We refer to the combination of a multi-dimensional binning model M with a measure μ to its product-bins as a *measure model* $M[\mu]$. The idea of precomputing and representing a *measure model* $M[\mu]$ so that we can quickly access $\mu(\beta)$ for any product-bin β is essentially the well-known notion of a *cube relational operator* (if all hierarchies in the model are all 1-level) or the more general *roll up cube relational operator* (if some hierarchies have 2 or more levels). Note that in practice, when precomputing such measure models, one does not expect to be able to retrieve the original records $A(\beta)$, but only its measure $\mu(\beta)$.

3.3.2 Nanocubes

In Nanocubes [47], the authors propose an efficient encoding of a measure model $M[\mu]$ with an additional special encoding for one *temporal* dimension. Nanocubes uses a pointer-based sparse data structure to represent the product-bins β that have at least one record associated with it, and tries to make every product-bin that yields the same set of records refer to the same memory location encoding its measure value. Conceptually, we can think of Nanocubes as an encoding to a mapping $\{\beta \mapsto \mu(\beta) : \beta \in \mathcal{B}, A(\beta) \neq \varnothing\}$. For the temporal dimension, the particular μ values are stored in Nanocubes as *summed area tables*:

$$\beta \mapsto ((t_1, v_1), (t_2, v_1 + v_2), \ldots, (t_p, v_1 + \ldots + v_p)), \tag{3.1}$$

where t_i's are all the *finest temporal bins* associated to the records in $A(\beta)$, they are sorted $t_i < t_{i+1}$, and v_i is the measure of $\mu(\beta, b_{\texttt{time}=t_i})$, i.e., product-bin with the added constraint in the time dimension. Note that by taking differences of values from two different indices of a summed area table one can quickly find the value of any query $(\beta, [t_a, t_b])$, where β is a product-bin (without the time dimension) and $[t_a, t_b]$ is the time interval of interest.

3.4 TopKube

A Nanocubes-like approach can efficiently retrieve a measure of interest for any pre-defined “bucket” (i.e., a product-bin plus a time interval). This capability can be handy for many applications, but is especially useful for interactive visualizations where each element presented on a screen (e.g., bar in a barchart, pixel in a heatmap) is associated with one of these “buckets” and encoded (e.g., bar length, pixel color) based on its value. However, suppose that, instead of simply accessing the measure associated with specific buckets, we are actually interested in identifying the top-k valued objects from a potentially large set of buckets. For example, “Who are the top-20 players that make the most shots from the right-hand 3-point corner of the basketball court?” (blue selection and ranking shown in Figure 3.1).

Since there is no ranking information encoded in a Nanocube, the only way to obtain such a top-20 rank is to find out, for each player associated with a shot in the selection, their total number of shots and report the top-20 players found. This computation takes time proportional to at least the number of players associated with the shots in the selection. While this computation in the case of NBA shots is not very expensive (only a few thousand players ever played in the NBA), there are interesting use cases, analogous to the *player-shot* case, where the number of “players” can be quite large. For instance, *project-commit* on GitHub (a cloud based project repository), *tag-photo* on Flickr (a cloud based photo repository), or *hashtag-post* on Twitter. In these cases the number of projects, tags, and hashtags are counted in millions instead of in thousands. The need to scan millions of objects to solve a single top-k query can be a hard hit in the latency budget of a fluid interactive experience.

TopKube is a data structure similar to a Nanocube: it encodes a *measure* in a multi-dimensional binning model, and, with this encoding, it allows the quick access of the measure’s value of any product-bin in the model. The main addition of a TopKube when compared to a Nanocube is that, in order to speed up top-k queries on one of its dimensions (e.g., top players by number of shots), a TopKube also includes ranking information in the encoding of that dimension.

The special dimension in a TopKube is one that could be modeled as yet another 1-level bin hierarchy, but that contains lots of bins (e.g., players in the

NBA example, or projects on GitHub, or tags on Flickr) and that we are interested in quickly accessing the top valued bins from this dimension with respect to the additive measure of interest on any multi-dimensional selection. We refer to this special dimension of a TopKube as its *key dimension*, and the bins in this dimension as *keys*. Note that efficiently retrieving ranks of top-k *keys* (and their respective values) for an arbitrary selection of product-bins is the main goal of our TopKube data structure. All dimensions in a TopKube, except for its *key dimension*, are represented in the same way as the (non-special) dimensions of a Nanocube: as nested bin-hierarchies. Nodes in the bin-hierarchy of a previous dimension point to a root bin of a bin-hierarchy in the next dimension until we get to the last special dimension (see Figure 2 of [47]). A path through the nested hierarchies down to the last and special dimension of a TopKube corresponds to a product-bin β on all dimensions except the key dimension.

To represent the *key dimension* information associated with a product-bin β, TopKube uses the following data:

$$\beta \mapsto \left\{q, v, \sigma, \sum v_i\right\}, \tag{3.2}$$

where $q = q_1 \dots q_p$, $v = v_1 \dots v_p$, and $\sigma = \sigma_1 \dots \sigma_p$ are arrays of equal length obeying the following semantics: q_i is the i-th smallest key that appears in β; v_i is the value of the measure of interest (e.g., occurrences) for key q_i in β; and σ_i represents index of the key with the i-th largest value in β. For example, the third highest values key in a specific β is given v_{σ_3} and corresponds to key q_{σ_3}. In addition to arrays q, v, σ, in order to quickly solve queries that contain no key constraints, we also store the measure of all records in β regardless of keys, i.e., $\mu(A(\beta))$. Since in all our applications we always assume linearity of our measures, this aggregate reduces to the sum of the values v in β.

In Figure 3.3, we show a concrete TopKube corresponding to the model shown on the top left part of the display. This TopKube consists of one spatial dimension (two level quad-tree hierarchy) and a key dimension. In this toy example, the keys of the key dimension are the letters `A`, `B`, and `C` and the measure is simply the number of occurrences of a letter in the corresponding product-bin. Note that since there is only one dimension outside of the key dimension in this example, a product-bin β corresponds exactly to one spatial bin. The TopKube data structure

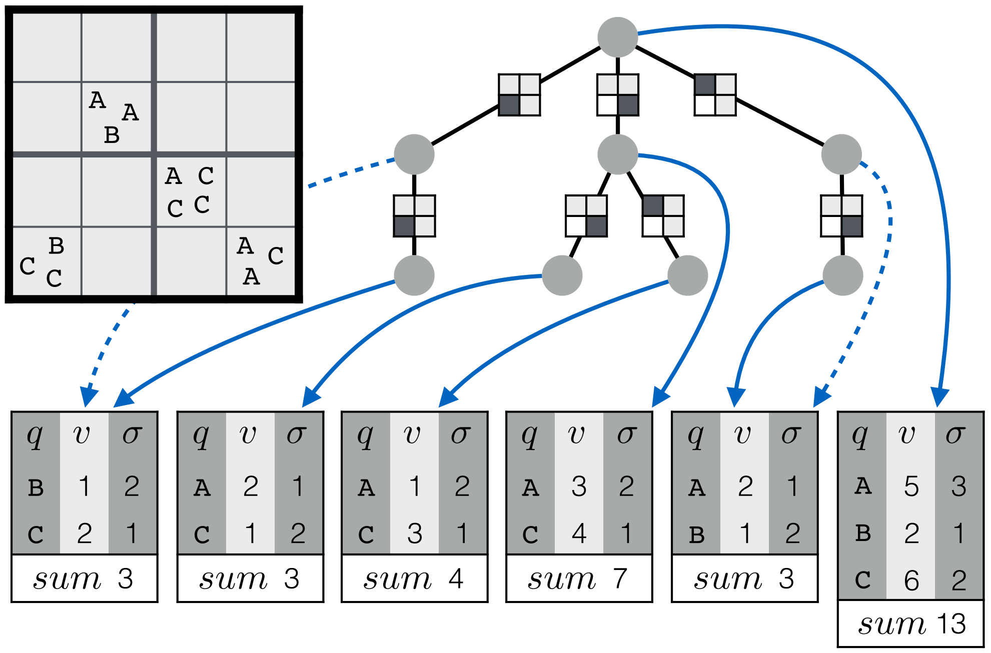


Figure 3.3: Concrete example of a TopKube with one spatial dimension and the special key-dimension for counting and ranking the event types: `A`, `B`, or `C`. The additional ranking information $(q, v, sigma)$ from Equation 3.2 is shown in the tables associated with each product-bin.

with the keys, counts, rank and total count are shown as tables in the bottom part of the figure. Note, for example, that the top valued key in the whole model is given by $q_{\sigma_1} = \texttt{C}$ and $v_{\sigma_1} = \texttt{6}$ in the right-most table which corresponds to the coarsest spatial bin.

With this encoding for the key dimension information of a product-bin, to find out if a given key exists in a product-bin, we can perform a binary search in the q array (logarithmic time in the length of the array), and to access the i-th top ranked key we perform two random accesses: first we retrieve σ_i and then q_{σ_i} or v_{σ_i} (both constant time).

As in a Nanocube, the size of a TopKube is proportional to its number of product-bins β plus the size of the encodings of the special dimension information associated with each of its product-bins. In the case of a Nanocube, this extra size per product-bin is the size of the summed area data from Equation 3.1, while in the case of TopKube, it is given by the size of the rank aware data-structure of Equation 3.2. Note that if a Nanocube and a TopKube have the same set of product-bins β and the number of time stamps and keys encoded in their respective special dimensions are comparable, the extra size cost of a TopKube compared to the similar Nanocube will be the rank arrays σ. This extra size cost of a TopKube represents a good trade-off if queries for interactive top-k keys are important for a given application. Another important remark with respect to the sizes of Nanocubes and TopKube is that in order to represent a Nanocube special temporal dimension into a TopKube dimension, we have to convert it into a conventional TopKube dimension (e.g., a binary tree where the leaves are timestamps: right side of Figure 3.2). This adds a multiplicative logarithmic term to the size of that dimension: while $O(n)$ in a Nanocube, it becomes $O(n \log n)$ in a TopKube. The advantage here is that now multiple temporal dimensions can be supported.

3.4.1 Top-K from Ranked Lists

The easiest top-k query for a TopKube happens when a single product-bin β in involved. Suppose a user wants the top ranked keys in a multi-dimensional selection without any constraints. This query boils down to the single coarsest product-bin β in the cube (formed by root bins in all dimensions). In this case, obtaining the top-k keys is the same as generating from β the list $(q_{\sigma_1}, v_{\sigma_1}), \ldots, (q_{\sigma_k}, v_{\sigma_k})$, and, clearly, it can be done in $O(k)$ steps. In general, though, this task is not that easy. The number of product-bins involved in the answer of a multi-dimensional selection is not one. For common spatial brushes, time intervals, categorical selections, the typical number of product-bins involved in a query ranges from tens up to a few thousand. For example, in Figure 3.2, we show a 624 bin selection in space and 3 bins in time which potentially means a 1,872 product-bin selection. In this case, the pre-stored ranks, or σ, we have for each product-bin should help speed up the top-k query, but is not as trivial as collecting top-k keys and values in $O(k)$ steps. Due to the lack of a consistent name in the literature, we refer to this problem as *Top-k from Ranked Lists* or TKR.

```
function SWEEP($\mathcal{L}$, $k$)
  $h \leftarrow []$
  for $i = 1$ to LENGTH($\mathcal{L}$) do
    if LENGTH($\mathcal{L}[i]$) $> 0$ then
      PUSHHEAP($h$, ($\mathcal{L}[i]$,1), LESSTHANID)
  $r \leftarrow []$
  $key \leftarrow$ null
  $sum \leftarrow 0$
  while LENGTH($h$) $> 0$ do
    $(\ell, i) \leftarrow$ POPHEAP($h$, LESSTHANID)
    if $key \neq \ell[i].key$ then
      INSERTK($r$,$k$,$key$,$sum$)
      $key \leftarrow \ell[i].key$
      $sum \leftarrow \ell[i].value$
    else
      $sum \leftarrow sum + \ell[i].value$
    if $i <$ LENGTH($\ell$) then
      PUSHHEAP($h$, ($\ell$,$i+1$), LESSTHANID)
  INSERTK($r$,$k$,$key$,$sum$)
  return $r$

procedure INSERTK($r$,$k$,$key$,$sum$)
  if $key \neq$ null then
    if LENGTH($r$) $< k$ then
      PUSHHEAP($r$, ($key, sum$), LESSTHANSUM)
    else if $r[1].value < sum$ then
      POPHEAP($r$, LESSTHANSUM)
      PUSHHEAP($r$, ($key, sum$), LESSTHANSUM)

function LESSTHANID(($\ell_1$,$i_1$), ($\ell_2$,$i_2$))
  return $\ell_1[i_1].key < \ell_2[i_2].key$

function LESSTHANSUM(($key_1, sum_1$), ($key_2, sum_2$))
  return $sum_1 < sum_2$
```

Figure 3.4: Pseudo-code for the Sweep Algorithm

3.4.2 Sweep Algorithm

Let us step back a bit. Suppose we do not store the ranking information, σ, in each product-bin. In other words, if we go back to a rank-unaware data structure, how can we solve the top-k keys problem? One way, which we refer to as the Naive Algorithm, is to traverse the key and value arrays (q and v in Eq. 3.2) of all the product-bins in the selection, and keep updating a dictionary structure of key-value pairs. We would increment the value of a key q_i already in the dictionary with the current value v_i found for that key in the current product-bin. Once we finish traversing all product-bins, we would sort the keys by their values and report the top-k ones. The Naive Algorithm is correct, but inefficient. It uses memory proportional to all the keys present in all lists of all product-bins in the selection, and this number might be much larger than k.

A more efficient way to find the top-k keys in the union of multiple product-bins $\beta_1 \dots \beta_m$ is shown in the pseudo-code listed in Figure 3.4. Assume in the pseudo-code descriptions that $\mathcal{L}$ is a list of m key-value-rank data structures corresponding to Eq. 3.2 of the m input product-bins. The idea is to create, from $\mathcal{L}$, a heap/priority queue where the product-bin with a current smallest key is on the top of the heap (Lines 3-5). If we keep popping the next smallest key (Line 10) and its value from all the lists, we will sweep all key-value pairs in key increasing order, and every time we get a larger key (Line 11), we can be sure that the total measure of the previous key was complete. Using this approach, we can maintain a result buffer of size k (Line 23) instead of a dictionary with all keys in the all lists. We will refer to this approach as the Sweep Algorithm. Note that this algorithm scans all keys of the product-bins β in the selection, as does the Naive Algorithm, but it does not need a potentially large buffer to solve the top-k problem. If we assume N is the sum of the number of keys in each input product-bin, it is easy to see that the worst case complexity of the Sweep Algorithm is $O(m \log m + N \log k + N \log m)$.

Although the top-k problem was not discussed in the original Nanocubes paper [47], the Sweep Algorithm can also be used to solve top-k queries there. Note also that Sweep Algorithm is a natural way to solve unique count queries in both TopKube and Nanocubes (how many unique keys are present in multi-dimensional selection).

3.4.3 Threshold Algorithm

The idea for adding the ranking information, σ, into a TopKube is that it can potentially reduce the number of steps needed to find the top-k keys compared to the number of steps Sweep Algorithm does. Instead of scanning all entries in all product-bins in our selection in key order, we would like to use the ranking information to scan first those keys with a higher chance of being in the top-k keys (i.e., larger partial values). The hope is that, by using such a strategy, a

```
 1: function TA($\mathcal{L}$, $k$)
 2:   $queues \leftarrow []$
 3:   $max \leftarrow 0$
 4:   for $i = 1$ to Length($\mathcal{L}$) do
 5:     if Length($\mathcal{L}[i]$) $> 0$ then
 6:       PushBack($queues$, ($\mathcal{L}[i]$,1))
 7:       $max \leftarrow$ max + ValueByRank($\mathcal{L}[i]$, 1)
 8:   $r \leftarrow []$
 9:   $processed \leftarrow \varnothing$
10:   $i \leftarrow 1$
11:   while $i \leq$Length($queues$) do
12:     $(\ell, rank) \leftarrow queues[i]$
13:     $(key, value) \leftarrow$ ValueByRank($\ell$,$rank$)
14:     if $rank <$Length($\ell$)
15:       $queues[i] \leftarrow (\ell, rank + 1)$
16:       $max \leftarrow max$+ValueByRank($\ell$, $rank + 1$)
17:       $i \leftarrow i + 1$
18:     else
19:       SwapBack($queues$,$i$)
20:       PopBack($queues$)
21:     $max \leftarrow max - value$
22:     if $key \notin processed$ then
23:       for $j = 1$ to Length($queues$) do
24:         $(\ell_j, \_) \leftarrow queues[j]$
25:         if $\ell \neq \ell_j$
26:           $value \leftarrow value$+ValueByKey($\ell_j$,$key$)
27:       InsertK($r$,$k$,$key$,$value$)
28:       if Length($r$)=$k$ and $max \leq r[1].value$ then
29:         break
30:       $processed \leftarrow processed \cup \{key\}$
31:     if $i >$ Length(queues) then
32:       $i \leftarrow 1$
33:   return $r$
```

Figure 3.5: Pseudo-code for the Threshold Algorithm. ValueByRank uses σ to retrieve the p-th largest value of $\mathcal{L}[i]$ in constant time. ValueByKey runs a binary search to access the value of a given key.

small partial scan and some bookkeeping would be enough to identify the top-k keys without a full scan. In fact, this outline of an algorithm is well-known in the computer science community (see pseudo-code in Figure 3.5): the famous Threshold Algorithm (TA) was described in [97]. Furthermore, TA was proven to be optimal in a strong sense: no other algorithm can access less data than it does and still obtain the correct answer. It essentially rotates through all ranked input lists (loop on Line 11) popping the highest value key in each of the lists (Line 13). For each popped key q_i, it goes through all the other lists (Loop on Line 23) binary searching for other key bins containing key q_i. Once the aggregated value of q_i is found, it inserts q_i and its final value into a running top-k result set (Line 27). Once the algorithm figures out that no other key can have a higher value than the current top-k keys, the computation is done (Line 29). The worst case complexity of the Threshold Algorithm is given by $O(m + Nm + N \log k)$. Note that the Nm term dominates this complexity and makes this worse than the one for Sweep Algorithm.

3.4.4 Hybrid Algorithm

Although Threshold Algorithm has the theoretical guarantees one would want, in practice we have observed that the instances of the TKR problem that show up in our use cases do not have the same characteristics as assumed in the explanations of TA that we reviewed. In those explanations, there was an implicit assumption that all m input lists in $\mathcal{L}$ contained the same set of keys. This is a natural assumption given the implicit application usually associated with TA: the m lists corresponded to m attribute-columns of a table with (mostly) non-zero entries. However, the instances of the TKR problem we observed in our use cases were sparse: one key q_i is present in only a small fraction of the m lists in the query selection. This sparseness introduces a wasteful step in the Threshold Algorithm: the loop on Line 23. Most of the binary searches in that loop will fail to find the key that we are searching for. A typical case we see in our instances of the TKR problem is that on average each key is present in less than 3% of the m lists in our query selections.

While the Threshold Algorithm can have wasted cycles trying to access entries for keys that do not exist, the Sweep Algorithm wastes no such cycles: all entries it accesses are present in the input lists. In order to get the best results in our experiments, we combined the strengths of TA (early termination using rank information) and the Sweep Algorithm (no wasted accesses on sparse instances) into a combined algorithm: the Hybrid Algorithm (pseudo-code shown in Figure 3.6).

The idea of the Hybrid Algorithm is simple: (1) raise the density of the input problem by running the Sweep Algorithm on the smallest (easiest) input lists, and (2) run the Threshold Algorithm on this denser equivalent problem. To make things more concrete, think about the 624 spatial bins in Figure 3.2. We typically expect the smaller squares in that spatial selection to have less data than larger squares.

```
 1: function HYBRID($\mathcal{L}$, $k$, $\theta$)
 2:   SORT($\mathcal{L}$, LESSTHANLENGTH)
 3:   $m \leftarrow$LENGTH($\mathcal{L}$)
 4:   $n \leftarrow$LENGTH($\mathcal{L}[m]$)
 5:   $entries \leftarrow n$
 6:   $i_{\text{split}} \leftarrow m$
 7:   for $i = m-1$ downto 1 do
 8:     $entries \leftarrow entries+$ LENGTH($\mathcal{L}[i]$)
 9:     $\theta_i \leftarrow entries/((m-i+1)*n)$
10:     if $\theta_i < \theta$ then
11:       break
12:     else
13:       $i_{\text{split}} \leftarrow i$
14:   if $i_{\text{split}} = 1$ then
15:     return TA($\mathcal{L}$, $k$)
16:   else if $i_{\text{split}} = m$ then
17:     return SWEEP($\mathcal{L}$, $k$)
18:   else
19:     $aux \leftarrow$ SWEEP($[\mathcal{L}[1], \ldots, \mathcal{L}[i_{\text{split}}]]$,$\infty$)
20:     $aux \leftarrow$ MAKERANK($aux$)
21:     return TA($[aux, \mathcal{L}[i_{\text{split}}+1], \ldots, \mathcal{L}[m]]$,$k$)
```

Figure 3.6: Pseudo-code for the Hybrid Algorithm

The idea would be to merge all the lists of the smaller squares to make the problem instance dense for the Threshold Algorithm. We formalize this process next.

The input parameter θ in the Hybrid Algorithm controls the *density level* of the input problem in order to be considered ready for the Threshold Algorithm. In other words, if, based on θ, our original input problem is too sparse for the Threshold Algorithm, we would like to merge the smaller i_{split} lists using the Sweep Algorithm (Line 19 of the Hybrid Algorithm), and then run an equivalent input denser problem through the Threshold Algorithm (Line 21). We define the *density* of a TKR instance to be N (i.e., sum of length of the q arrays in all product-bins of the selection) number of entries in all input lists, divided by the actual number of distinct keys (if we do the union of all m sets of keys) multiplied by m (the length of $\mathcal{L}$). Obviously, to compute this density one needs to find the number of keys after merging all the keys in arrays q for all m lists, which is an inefficient process that requires scanning all entries in all lists. To keep things computationally simple, and still correlated with the density notion, we define the *density level* θ as an upper bound for the actual density where we replace number of keys in the union of all m arrays by the length of the largest array q from the m product-bins (Lines 7-13).

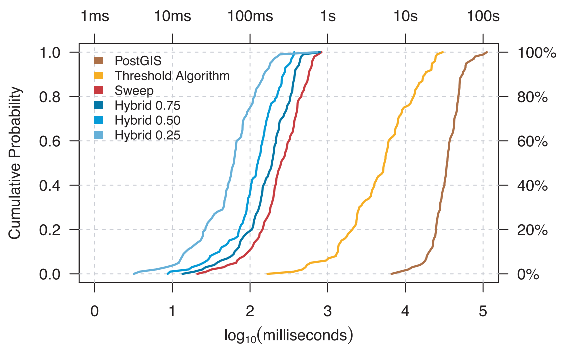


Figure 3.7: Empirical cumulative distributions of the time to retrieve the top-32 valued keys for 100 spatiotemporal queries on the Twitter dataset. Speedup potential of Hybrid versus Sweep, Threshold, and PostGIS.

3.5 Experimental Results

Our system was implemented using a distributed client-server architecture. The TopKube server program was implemented in C++ and provides a query API through HTTP. This enables flexibility: it can serve various client types ranging from desktop to mobile applications. All rendering in this work was done on client programs. We implemented a portable browser-based client using HTML5, D3, and WebGL as well as an OpenGL based C++ client for more native performance. The timing experiments relative to back-end performance in this chapter ran on a 64 core AMD Opteron with 2.3 GHZ CPU and 512 GB of RAM.

3.5.1 Performance

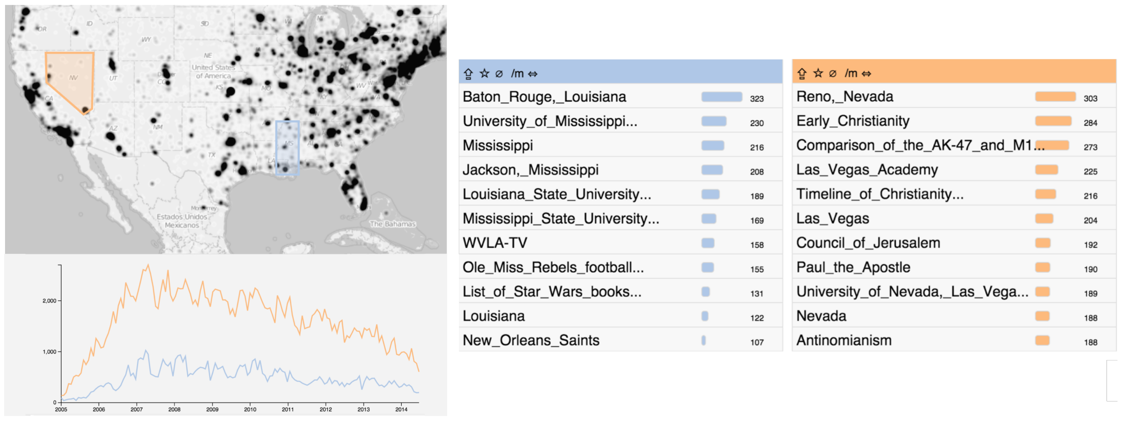
To determine which of the previously described algorithms works best when solving top-k queries, we conducted an initial evaluation using the Microblogs dataset, which is the most challenging because it has the most keys (4.7M). The first experiment consisted of collecting 100 spatiotemporal selections ranging from large geospatial areas (continents) to smaller regions (cities) combined with time interval selections ranging from multiple weeks to a few hours. Next, we retrieved the top-32 valued keys in each of the 100 selections with the different methods we describe in Section 3.4. In addition to SWEEP, THRESHOLD, and HYBRID, we also included PostGIS in this experiment. PostGIS is the most popular open source GIS package that can solve the problem that we were targeting in this work. It is the de facto spatial database in our opinion, which is why we chose to compare our techniques to it. We configured PostGIS according to its official documentation for a dataset containing key, latitude, longitude, and timestamp.

In Figure 3.7 we present the results of our first experiment in the form of cumulative distributions: what percentage of the 100 spatiotemporal queries we could retrieve the top-32 keys in less than t time units. All results were exactly the same for all the methods tested including PostGIS. We are able to see that the Hybrid Algorithm with varying θ thresholds had query times consistently smaller than both TA and SWEEP. This fact confirmed our hypothesis that we can accelerate top-k queries by adding rank information to the index. Although this fact seems obvious, this study shows that a natural use of rank information as done by TA does not yield a speedup. Only a combination of the strengths of TA and SWEEP illustrated by the HYBRID approach gave the speedup we expected. It is worth noting, however, that there was a steep increase in query times for HYBRID on the most difficult problems (as cumulative probability approached 1), which suggests that a better balance between SWEEP and TA was possible. In Section 3.7 we perform a more thorough experiment to understand the behavior of our top-k methods.

3.6 Case Studies

The datasets used in our case studies are freely available and allow the extraction of geotagged keywords: articles on Wikipedia, tags on Flickr, projects on GitHub, and hashtags on Microblogs. The number of records (in millions) for these four datasets are respectively: 112M, 84M, 58M, and 40M. The number of keywords (in millions) are respectively: 3.0M, 1.6M, 1.5M, and 4.7M; thus although Wikipedia has the most records, Microblogs has the most unique keywords. The keywords were then used as *keys* in the construction of our TopKube data structure.

Wikipedia. The Wikipedia English dump datasets [104] contains edit history for every article since its creation in 2005. Anonymous edits contain the IP information of the user, which we used to trace their location. The final dataset, with geographical information, contains more than 112 million edits of over three million articles. Figure 3.8 presents a visualization of the dataset using TopKube. It is interesting to see that even though Nevada is not considered a state with a



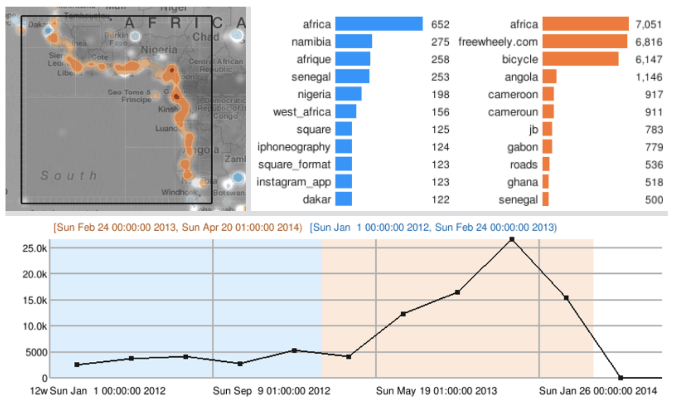
Baton_Rouge,_Louisiana	323
University_of_Mississippi...	230
Mississippi	216
Jackson,_Mississippi	208
Louisiana_State_University...	189
Mississippi_State_University...	169
WVLA-TV	158
Ole_Miss_Rebels_football...	155
List_of_Star_Wars_books...	131
Louisiana	122
New_Orleans_Saints	107

Reno,_Nevada	303
Early_Christianity	284
Comparison_of_the_AK-47_and_M1...	273
Las_Vegas_Academy	225
Timeline_of_Christianity...	216
Las_Vegas	204
Council_of_Jerusalem	192
Paul_the_Apostle	190
University_of_Nevada,_Las_Vega...	189
Nevada	188
Antinomianism	188

Figure 3.8: Comparing the top edited articles in Nevada and Mississippi.

high percentage of religious people, religious articles are among the highest ranked. On the other hand, Mississippi, considered one of the most religious states in the U.S., does not have a single article related to religion among the top-20.

Flickr. The Yahoo! Flickr Creative Commons dataset [19] contains 100 million public Flickr photos and videos, with each record containing a set of user tags and geographical information. The dataset contains 84 million geolocated tags (1.57 million unique ones). Figure 3.9 shows how exploration can be used to gain insight of unusual patterns in the data along the West Coast of Africa.

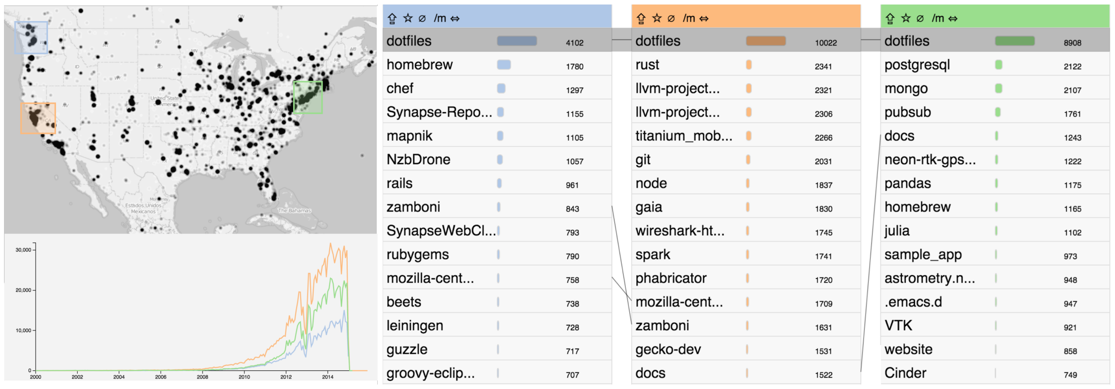


africa	652
namibia	275
afrique	258
senegal	253
nigeria	198
west_africa	156
square	125
iphoneography	124
square_format	123
instagram_app	123
dakar	122

africa	7,051
freewheely.com	6,816
bicycle	6,147
angola	1,146
cameroon	917
cameroun	911
jb	783
gabon	779
roads	536
ghana	518
senegal	500

Figure 3.9: Geolocated Flickr tags in Africa: the unusual activity on the west coast are from photos taken during a bike trip.

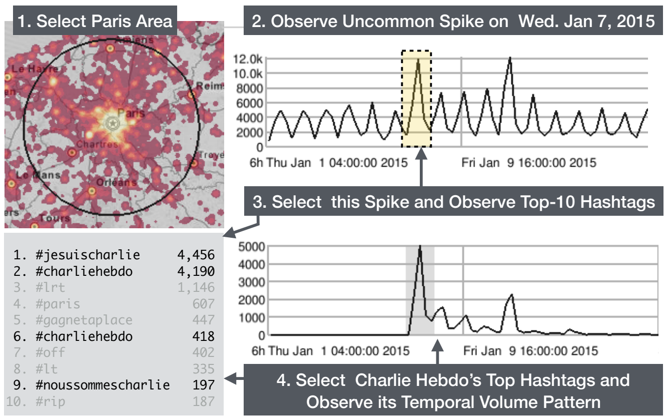
By highlighting the region, we can see that there were an unusual spike of activity during a few days in January. We create two different brushes in the timeseries: a blue one covering the low activity days, and an orange one covering the high activity days. We can see that the high activity spike is mostly due to photos tagged with *freewheely.com* and *bicycle*, which were taken by a Flickr user during his bike trip.



dotfiles	4102	dotfiles	10022	dotfiles	8908
homebrew	1780	rust	2341	postgresql	2122
chef	1297	llvm-project...	2321	mongo	2107
Synapse-Repo...	1155	llvm-project...	2306	pubsub	1761
mapnik	1105	titanium_mob...	2266	docs	1243
NzbDrone	1057	git	2031	neon-rtk-gps...	1222
rails	961	node	1837	pandas	1175
zamboni	843	gaia	1830	homebrew	1165
SynapseWebCl...	793	wireshark-ht...	1745	julia	1102
rubygems	790	spark	1741	sample_app	973
mozilla-cent...	758	phabricator	1720	astrometry.n...	948
beets	738	mozilla-cent...	1709	.emacs.d	947
leiningen	728	zamboni	1631	VTK	921
guzzle	717	gecko-dev	1531	website	858
groovy-eclip...	707	docs	1522	Cinder	749

Figure 3.11: GitHub projects with most commits in three large urban centers.

Twitter. This dataset is comprised of publicly available geo-tagged Twitter entries. From each post, we extracted the latitude, longitude, and hashtags from the blog. We can use TopKube to explore the most popular hashtags in order to understand how trending topics vary over time and in a given region. Figure 3.10 presents a sequence of exploration steps within January 2015 records. First we select a geographical area around Paris and find out an unusual Wednesday peak (Jan. 7) in the volume of hashtags. By selecting this peak we quickly find evidence of the event that caused the volume spike by inspecting the top-10 hashtags in the current selection (i.e., Paris and Jan 7). The event in question was the terrorist attack at the Charlie Hebdo headquarters. To understand how the hashtags created for this event at the day of the attack faded in time, we further constrain our selection to just the hashtags related to the terrorist attack and see that those fade almost completely (relative to event day) after one week of the attack.


1. Select Paris Area
2. Observe Uncommon Spike on Wed. Jan 7, 2015
3. Select this Spike and Observe Top-10 Hashtags
4. Select Charlie Hebdo's Top Hashtags and Observe its Temporal Volume Pattern
Le Havre
Reims
Paris
Chartres
Troye
Le Mans
Orléans
Tours
12.0k
10.0k
8000
6000
4000
2000
0
6h Thu Jan 1 04:00:00 2015
Fri Jan 9 16:00:00 2015
5000
4000
3000
2000
1000
0
6h Thu Jan 1 04:00:00 2015
Fri Jan 9 16:00:00 2015

1.	#jesuischarlie	4,456
2.	#charliehebdo	4,190
3.	#lrt	1,146
4.	#paris	607
5.	#gagnetaplace	447
6.	#charliehebdo	418
7.	#off	402
8.	#lt	335
9.	#noussommescharlie	197
10.	#rip	187

Figure 3.10: Twitter exploration using TopKube: a temporal perspective of the top hashtags related to the Charlie Hebdo terrorist attack in Paris.

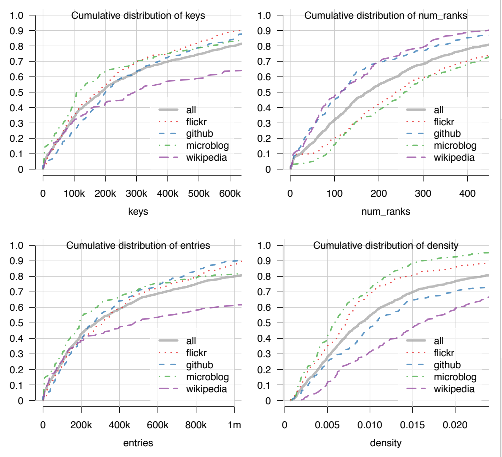
Github. The GitHub dataset was first made available by Gousios [105] and contains all events from the GitHub public event time line. We were able to obtain information on more than 58 million commits for roughly 1.5 million projects. Each commit was geolocated based on the location of the user responsible for the action. Figure 3.11 presents a visualization with the top-k projects of three large urban centers. The only common project among all three regions is *dotfiles*, a project for sharing customized environment files on Unix-based operating systems. It is also interesting to notice how *llvm* and related projects (such as *clang*), are very popular in California, but not elsewhere. This shows a highly diversified open source community across the United States.

3.7 TopKube-Benchmark

As illustrated in the previous examples, the main use case that drove the development of TopKube was to provide an interactive visualization front-end to quickly access top-k "terms" for arbitrary spatiotemporal selections. Although we observe significant speedups using the Hybrid Algorithm (e.g., $\theta = 0.25$ in Figure 3.7) compared to other techniques, we believe in further improvements. To assess how different top-k algorithms (the ones shown here and future ones) perform in rank merging problems on datasets similar to the ones we collected for this work, we created the TopKube-Benchmark.

3.7.1 Benchmark Characteristics

The TopKube-Benchmark consists of one thousand TKR problems. Each problem consists of a list of *ranks*, where each rank is defined by a list of key-value pairs and the associated ordering information, σ, as shown in Equation 3.2. The goal is to, given a value k, find the top-k keys and their aggregated value from a consolidated rank of the multiple input ranks (note that this problem does not require explicitly finding the total consolidated rank). Each of the four datasets (i.e., Flickr, GitHub, Microblog, and Wikipedia) contributed equally with two hundred and fifty problems for the the TopKube-Benchmark. These problems were collected during interactive exploratory sessions using these four datasets In Figure 3.12, we present the distribution of four characteristics of the problems in the benchmark: (1) number of keys; (2) number of ranks; (3) number of entries; and (4) density.



var.	dataset	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
keys	all	7.0E+00	1.8E+04	5.5E+04	8.7E+04	1.3E+05	1.9E+05	2.7E+05	4.0E+05	6.1E+05	1.1E+06	4.7E+06
	flickr	1.4E+03	2.2E+04	5.4E+04	7.9E+04	1.2E+05	1.8E+05	2.3E+05	3.0E+05	4.7E+05	6.3E+05	1.6E+06
	github	7.0E+00	4.2E+04	7.9E+04	1.2E+05	1.7E+05	2.0E+05	2.6E+05	3.7E+05	5.2E+05	7.1E+05	1.5E+06
	microblog	4.2E+02	4.1E+03	3.4E+04	6.6E+04	9.3E+04	1.1E+05	1.8E+05	3.0E+05	5.4E+05	1.1E+06	4.7E+06
	wikipedia	1.5E+03	2.1E+04	5.7E+04	9.4E+04	1.6E+05	3.0E+05	5.2E+05	9.4E+05	1.2E+06	1.7E+06	3.0E+06
num_ranks	all	1.0E+00	2.4E+01	6.0E+01	9.2E+01	1.3E+02	1.7E+02	2.4E+02	3.1E+02	4.3E+02	8.6E+02	4.3E+03
	flickr	1.0E+00	5.3E+01	1.0E+02	1.4E+02	1.9E+02	2.4E+02	3.1E+02	4.0E+02	6.7E+02	1.1E+03	4.3E+03
	github	1.0E+00	1.5E+01	3.2E+01	5.8E+01	8.5E+01	1.2E+02	1.4E+02	2.1E+02	3.3E+02	5.4E+02	2.6E+03
	microblog	1.0E+00	8.4E+01	1.2E+02	1.5E+02	2.1E+02	2.6E+02	3.1E+02	4.3E+02	6.4E+02	1.1E+03	2.5E+03
	wikipedia	1.0E+00	8.9E+00	3.8E+01	5.7E+01	7.4E+01	1.1E+02	1.6E+02	2.0E+02	3.1E+02	4.4E+02	3.0E+03
entries	all	9.0E+00	2.5E+04	7.4E+04	1.2E+05	1.9E+05	2.7E+05	4.2E+05	6.4E+05	1.0E+06	2.1E+06	2.0E+07
	flickr	1.9E+03	3.3E+04	8.8E+04	1.3E+05	2.0E+05	3.1E+05	4.2E+05	5.4E+05	8.0E+05	1.0E+06	2.0E+06
	github	9.0E+00	5.0E+04	9.9E+04	1.5E+05	2.0E+05	2.7E+05	3.7E+05	5.0E+05	7.3E+05	1.0E+06	2.3E+06
	microblog	5.4E+02	5.5E+03	5.0E+04	1.0E+05	1.4E+05	1.9E+05	2.8E+05	4.8E+05	8.4E+05	1.7E+06	7.0E+06
	wikipedia	1.6E+03	2.4E+04	7.0E+04	1.2E+05	2.2E+05	4.8E+05	9.0E+05	2.2E+06	3.2E+06	4.8E+06	2.0E+07
density	all	6.9E-04	2.3E-03	3.9E-03	5.4E-03	7.0E-03	8.9E-03	1.1E-02	1.5E-02	2.3E-02	5.1E-02	1.0E+00
	flickr	6.9E-04	2.0E-03	2.9E-03	4.4E-03	6.1E-03	7.2E-03	8.2E-03	1.0E-02	1.4E-02	2.8E-02	1.0E+00
	github	9.1E-04	2.8E-03	4.3E-03	6.8E-03	8.7E-03	1.1E-02	1.4E-02	2.0E-02	3.4E-02	8.0E-02	1.0E+00
	microblog	7.7E-04	1.6E-03	2.6E-03	3.8E-03	5.0E-03	5.9E-03	7.1E-03	9.7E-03	1.2E-02	1.6E-02	1.0E+00
	wikipedia	2.0E-03	4.9E-03	6.5E-03	9.8E-03	1.3E-02	1.6E-02	2.1E-02	2.8E-02	3.7E-02	2.2E-01	1.0E+00

Figure 3.12: Characteristics of the TKR problems in the TopKube-Benchmark. Left: we plot the cumulative distributions for the number of keys, ranks, entries, and density up to the 0.8 quantile (or 80th percentile). Right: we list the actual values of the percentiles for these distributions.

The number of keys of a problem is simply the union of the keys present in each rank. The number of ranks is the number of lists (of key values) from the selection. The number of entries is the sum of the sizes of the ranks (i.e., the total number of keys in all ranks). Note that number of entries should be larger than the number of keys since the same key is usually present in more than one rank. Finally, the density is simply the number of entries divided by the product of number of ranks and number of keys. If a problem has density one, each key is present in all ranks.

If we follow the overall thick solid gray line in the keys plot (Figure 3.12, top left), we notice that fewer than 40% of the problems involved fewer than 100k keys, which means that most problems (more than 60%) involved 100k keys or more. If we check the table entry in row *keys/all* and column *90%* from the table in that figure, we see that more than 10% of the problems involved 1.1 million keys or more. So, given that these problem instances were collected from natural visual interactions with the data, it is clear that large TKR problems can show up at exploration time: a challenging problem for interactivity. In terms of the number of ranks, we see that more than 50% of the problems have 170 or more ranks to be processed (row *num_ranks/all*, column *50%*), and in 10% of the cases we had 860 ranks or more (lots of non-empty product-bins being hit by the multi-dimensional selection). In terms of number of entries, we see that 20% of the problems had more than 1 million entries (row *entries/all*, column *80%*). Perhaps the most important observation of the problems in the TopKube-Benchmark comes from their density: the problems are really sparse (worst-case scenario for TA). If we consider 100 ranks in a problem and a density of 0.051 (90% of the problems have density 0.051 or less: see row *density/all*, column *90%*), on average we will have one key present in only 5.1 of the 100 ranks. These real-world, interactive explorations clearly demonstrate the sparsity of our inputs to the TKR problem, and that the binary searches on Line 23 in TA are largely wasted effort.

From the characteristics of the four datasets, we know that spatially the Microblog and Flickr datasets involve more spatial bins than the Wikipedia and GitHub datasets. The reason for this is simply that both Wikipedia and GitHub datasets were obtained by geocoding the IP address of the device associated with an article edit or project commit which induces a more constrained set of locations when compared to GPS locates from devices used for posts on Flickr and Twitter. This fact explains the distribution shown in the number of ranks plot in Figure 3.12: more product-bins are involved in the spatiotemporal selections for Flickr and Twitter than Wikipedia and GitHub. Given the similar sparse nature of the TKR problems that we see on these four datasets (see x-axis of the density plot in Figure 3.12), we focus the rest of our analysis here on the entire set of one thousand problems without splitting them by source.

3.7.2 Benchmark Performance

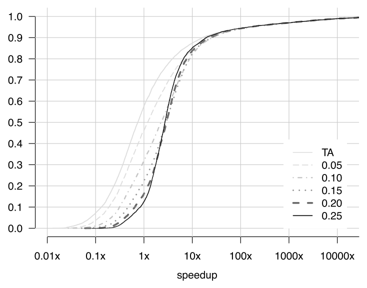
To assess the performance of the Sweep Algorithm, the Threshold Algorithm, and the Hybrid Algorithm, we ran each of the algorithms on the one thousand problems of the TopKube-Benchmark for $k = 5, 10, 20, 40, 80, 160, 320$, for a total of seven thousand runs for each algorithm. We ran the Hybrid Algorithm with the threshold θ varying from 0.05 to 0.95 by increments of 0.05. So, for each problem and each k, we ran the Sweep Algorithm and the Threshold Algorithm each once, and the Hybrid Algorithm nineteen times (one for each θ): a total of 21 different algorithmic recipes to find the top-k terms. The (percentiles of the) distributions of the latency (i.e., time to solve) for each of the seven thousand TKR instances by each of the 21 strategies are shown in Figure 3.13.

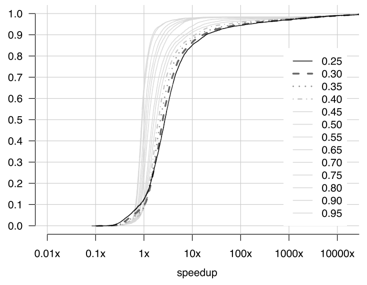
θ \ %	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
TA	1.32E−03	5.73E−03	1.70E−02	3.94E−02	7.75E−02	1.48E−01	3.02E−01	7.21E−01	2.43E+00	1.68E+02
0.05	1.25E−03	4.66E−03	1.17E−02	2.59E−02	5.04E−02	9.57E−02	1.83E−01	4.14E−01	1.22E+00	6.71E+01
0.10	1.08E−03	3.80E−03	8.34E−03	1.64E−02	3.03E−02	5.48E−02	1.04E−01	2.02E−01	5.48E−01	2.34E+01
0.15	1.00E−03	3.38E−03	7.36E−03	1.41E−02	2.53E−02	4.42E−02	7.65E−02	1.43E−01	3.51E−01	1.06E+01
0.20	9.79E−04	3.40E−03	7.31E−03	1.41E−02	2.37E−02	4.18E−02	7.20E−02	1.29E−01	2.97E−01	6.43E+00
0.25	9.80E−04	3.48E−03	7.63E−03	1.43E−02	2.44E−02	4.15E−02	7.22E−02	1.29E−01	3.00E−01	4.76E+00
0.30	9.85E−04	3.69E−03	8.21E−03	1.54E−02	2.63E−02	4.41E−02	7.36E−02	1.33E−01	3.19E−01	4.15E+00
0.35	1.10E−03	3.92E−03	9.19E−03	1.67E−02	2.89E−02	4.81E−02	7.76E−02	1.47E−01	3.53E−01	4.07E+00
0.40	1.23E−03	4.41E−03	9.95E−03	1.88E−02	3.21E−02	5.38E−02	8.51E−02	1.65E−01	3.94E−01	4.24E+00
0.45	1.45E−03	4.99E−03	1.14E−02	2.13E−02	3.58E−02	5.86E−02	9.41E−02	1.80E−01	4.17E−01	4.51E+00
0.50	1.78E−03	5.65E−03	1.27E−02	2.38E−02	3.97E−02	6.54E−02	1.03E−01	1.96E−01	4.54E−01	4.83E+00
0.55	2.23E−03	6.94E−03	1.45E−02	2.61E−02	4.47E−02	7.03E−02	1.14E−01	2.08E−01	4.82E−01	5.09E+00
0.60	2.47E−03	7.89E−03	1.67E−02	2.83E−02	4.97E−02	7.90E−02	1.29E−01	2.26E−01	5.29E−01	5.40E+00
0.65	2.99E−03	8.94E−03	1.90E−02	3.23E−02	5.59E−02	8.68E−02	1.39E−01	2.42E−01	5.79E−01	5.72E+00
0.70	3.16E−03	1.00E−02	2.03E−02	3.53E−02	6.17E−02	9.58E−02	1.51E−01	2.67E−01	6.09E−01	6.08E+00
0.75	3.43E−03	1.07E−02	2.27E−02	4.01E−02	6.91E−02	1.06E−01	1.67E−01	2.94E−01	6.47E−01	6.21E+00
0.80	3.63E−03	1.20E−02	2.67E−02	4.55E−02	7.71E−02	1.16E−01	1.80E−01	3.10E−01	6.75E−01	6.48E+00
0.85	3.84E−03	1.41E−02	2.94E−02	5.11E−02	8.48E−02	1.25E−01	1.98E−01	3.22E−01	7.01E−01	6.47E+00
0.90	4.01E−03	1.45E−02	3.06E−02	5.38E−02	8.91E−02	1.36E−01	2.06E−01	3.38E−01	7.35E−01	6.64E+00
0.95	3.98E−03	1.51E−02	3.17E−02	5.53E−02	9.13E−02	1.41E−01	2.18E−01	3.48E−01	7.52E−01	6.64E+00
Sweep	7.66E−03	2.17E−02	3.73E−02	5.84E−02	8.58E−02	1.28E−01	1.91E−01	3.03E−01	6.21E−01	6.63E+00

0.5 ms. <	1 ms. <	5 ms. <	10 ms. <	50 ms. <	100 ms. <	500 ms. <	1 s. <	5 s. <	10 s.

Figure 3.13: Latency distribution (percentiles) of twenty one different strategies (1 x TA, 19 x Hybrid, 1 x Sweep) when solving the one thousand TopKube-Benchmark problems for seven different values of $k = 5, 10, 20, 40, 80, 160, 320$. Darker blue shades indicate smaller latencies; darker red shades indicate larger latencies.

We assume that less than 0.1 seconds latency is the appropriate level for fluid interactivity: blue color tones in Figure 3.13 correspond to latencies that are less than 0.1 seconds, while red tones represent high latencies (≥ 0.1s.). This table contains clear evidence that the Hybrid strategy can improve on the latency of the two extreme strategies: sweep (which consolidates a full rank before generating top-k) and the threshold strategy which chases a proof of the top-k terms directly from all ranks of the input problem. Note that for values of θ between 0.15 and 0.45, the shades of blue have the widest reach: 70% of the problems had less than 100 ms latency, while for the sweep strategy 40% of the problems had 128 ms latency or more (column 60%).





θ \ %	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
TA	1.31E−01	2.38E−01	3.64E−01	5.16E−01	7.46E−01	1.11E+00	1.84E+00	3.90E+00	1.63E+01	1.52E+05
0.05	1.88E−01	3.27E−01	4.99E−01	7.58E−01	1.19E+00	1.92E+00	3.26E+00	6.60E+00	2.14E+01	1.43E+05
0.10	3.11E−01	5.52E−01	9.14E−01	1.46E+00	2.24E+00	3.47E+00	5.32E+00	8.81E+00	2.66E+01	1.52E+05
0.15	4.74E−01	8.88E−01	1.45E+00	2.11E+00	2.88E+00	3.95E+00	5.56E+00	8.62E+00	2.52E+01	1.37E+05
0.20	6.48E−01	1.21E+00	1.73E+00	2.22E+00	2.94E+00	3.74E+00	4.97E+00	7.66E+00	2.29E+01	1.37E+05
0.25	8.42E−01	1.37E+00	1.76E+00	2.21E+00	2.71E+00	3.34E+00	4.34E+00	6.59E+00	1.93E+01	1.43E+05
0.30	1.02E+00	1.40E+00	1.72E+00	2.08E+00	2.45E+00	2.99E+00	3.84E+00	5.73E+00	1.69E+01	1.52E+05
0.35	1.06E+00	1.36E+00	1.63E+00	1.89E+00	2.19E+00	2.65E+00	3.36E+00	4.83E+00	1.38E+01	1.43E+05
0.40	1.05E+00	1.30E+00	1.51E+00	1.72E+00	1.99E+00	2.34E+00	2.91E+00	4.17E+00	1.15E+01	1.37E+05
0.45	1.02E+00	1.22E+00	1.40E+00	1.58E+00	1.81E+00	2.08E+00	2.56E+00	3.53E+00	8.35E+00	1.37E+05
0.50	9.98E−01	1.15E+00	1.30E+00	1.44E+00	1.63E+00	1.87E+00	2.29E+00	3.08E+00	7.11E+00	1.37E+05
0.55	9.59E−01	1.09E+00	1.22E+00	1.34E+00	1.50E+00	1.72E+00	2.03E+00	2.71E+00	5.61E+00	1.52E+05
0.60	9.22E−01	1.04E+00	1.14E+00	1.25E+00	1.38E+00	1.57E+00	1.85E+00	2.34E+00	4.05E+00	1.43E+05
0.65	8.73E−01	9.76E−01	1.07E+00	1.16E+00	1.28E+00	1.42E+00	1.66E+00	2.14E+00	3.40E+00	1.35E+05
0.70	8.37E−01	9.29E−01	1.01E+00	1.09E+00	1.17E+00	1.30E+00	1.49E+00	1.83E+00	2.88E+00	1.35E+05
0.75	7.92E−01	8.79E−01	9.48E−01	1.01E+00	1.09E+00	1.20E+00	1.37E+00	1.68E+00	2.46E+00	1.04E+05
0.80	7.50E−01	8.30E−01	8.94E−01	9.55E−01	1.03E+00	1.12E+00	1.24E+00	1.45E+00	2.09E+00	9.34E+04
0.85	7.16E−01	7.94E−01	8.51E−01	9.05E−01	9.66E−01	1.04E+00	1.15E+00	1.33E+00	1.73E+00	1.35E+05
0.90	6.95E−01	7.62E−01	8.17E−01	8.69E−01	9.21E−01	9.86E−01	1.08E+00	1.24E+00	1.62E+00	1.29E+05
0.95	6.69E−01	7.39E−01	7.94E−01	8.42E−01	8.91E−01	9.51E−01	1.04E+00	1.19E+00	1.52E+00	1.43E+05

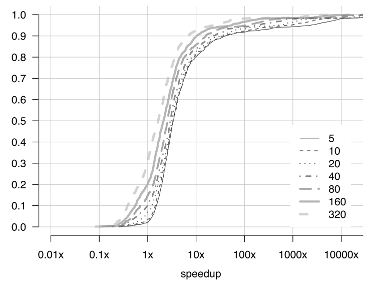
1/100x <	1/10x <	1/5x <	1/2x <	1/1.5x <	1/1.1x <	1.1x <	1.5x <	2x <	5x <	10x <	100x

Figure 3.14: Speedup distribution of the Threshold and Hybrid algorithms ($0 < \theta < 1$) over Sweep algorithm for all problems in the TopKube-Benchmark.

To more deeply explore the speedup of the Hybrid Algorithm over the Sweep Algorithm on the TopKube-Benchmark problems, we divide the query times (i.e., latency) for each problem instance i: $speedup_i^\theta = \frac{sweep\_time_i}{hybrid\_time_i^\theta}$. If it takes two times more for the sweep strategy to solve an instance compared to the hybrid strategy, we say there was $2\times$ speedup. On the other hand, if the sweep strategy takes half the time, we say there was $1/2\times$ speedup or, equivalently, a $2\times$ slow down.

We consider Sweep as the baseline approach, and we want to understand how Threshold and Hybrid compare to it. Figure 3.14 shows the cumulative distribution of the speed-up (or slow-down if less than 1.0) for different runs of Hybrid with threshold θ varying by 0.05 from 0 to 0.95. Note that Threshold is simply Hybrid with $\theta = 0$. Slow downs of 10% or more are colored in shades of red, and speed ups of 10% or more are colored in shades of blue. For $\theta = 0.25$, 80% of the problems had a speed up of 37% or more (column 20%); 70% of the instances had a speed up of 76% or more (column 30%); and 10% of the instances had a speedup of an order of magnitude (at least 19.3×).

It is also clear that our choice of θ impacts how well the Hybrid Algorithm performs. For $\theta = 0.95$, 50% of the latencies were 10% worse than the sweep strategy.



k \ %	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
5	1.52E+00	1.97E+00	2.34E+00	2.83E+00	3.40E+00	4.11E+00	5.67E+00	9.81E+00	4.07E+01	1.43E+05
10	1.44E+00	1.83E+00	2.27E+00	2.71E+00	3.34E+00	4.22E+00	5.57E+00	9.02E+00	3.74E+01	1.35E+05
20	1.24E+00	1.63E+00	2.09E+00	2.62E+00	3.16E+00	3.90E+00	5.15E+00	8.37E+00	3.18E+01	8.84E+04
40	1.01E+00	1.48E+00	1.90E+00	2.45E+00	2.93E+00	3.53E+00	4.63E+00	6.86E+00	2.06E+01	4.60E+04
80	7.30E−01	1.31E+00	1.68E+00	2.13E+00	2.58E+00	3.08E+00	3.90E+00	5.81E+00	1.61E+01	2.74E+04
160	5.54E−01	9.96E−01	1.36E+00	1.65E+00	2.13E+00	2.69E+00	3.39E+00	4.70E+00	1.04E+01	1.43E+04
320	3.86E−01	6.80E−01	9.73E−01	1.24E+00	1.64E+00	2.11E+00	2.65E+00	3.61E+00	6.72E+00	5.57E+03

1/100x <	1/10x <	1/5x <	1/2x <	1/1.5x <	1/1.1x <	1.1x <	1.5x <	2x <	5x <	10x <	100x

Figure 3.15: Distribution of the speedup of the Hybrid Algorithm ($\theta = 0.25$) over the Sweep Algorithm broken down by k.

In general, $\theta = 0.25$ performs very well overall and tends to be our default selection; however, $\theta = 0.20$ is arguably just as efficient and even beats $\theta = 0.25$ between 40% - 90%.

As can be seen in Figure 3.12, we explicitly included some extreme problem instances into TopKube-Benchmark: problems with a single rank and very few entries/keys, or conversely thousands of ranks and millions of entries/keys. These problems show up in the 0 and 100 percentile columns of that table, as well as the 100 percentile column of the colored distribution tables shown. We do not place undue emphasis on those columns: a speed up of one hundred thousand times ($\theta = 0.25$ column 100%) is not representative of the typical cases, while the speedups up to 90% are more typical.

3.7.3 Speedup Relative to k

In Figure 3.15, we present the speedup over Sweep by Hybrid with $\theta = 0.25$, for the different values of k. The blue and red shading follows the same speedup encoding as in Figure 3.14. As expected, a higher value of k demands more computation to find the top-k terms in the consolidated rank. For $k = 5$, 90% of the instances were solved at least 52% faster than the sweep approach (row 5, column 10%); for $k <= 20$, 80% had a 2× speedup; and for $k = 320$, 60% of the instances had a 24% speed up. We argue that a value of $k \leq 100$ is appropriate for exploration of top terms at interactive rates. Going deeper (i.e., larger k's) on an investigation could use more computational resources and a full sweep based consolidated rank could be used there.

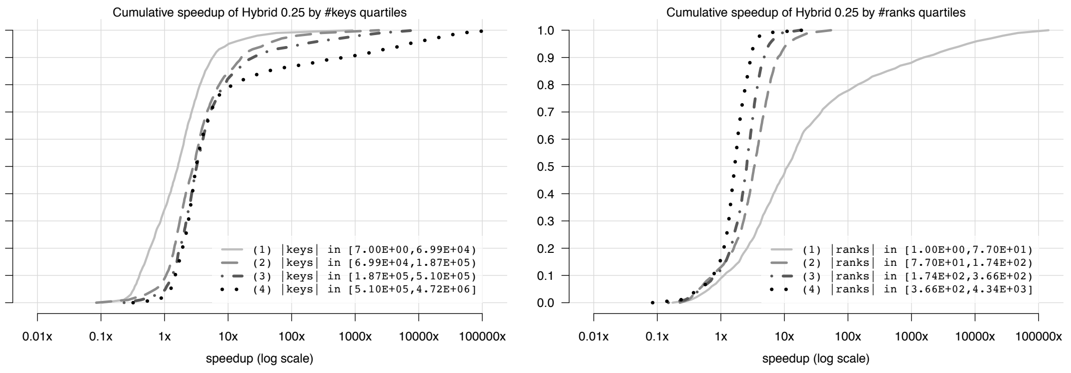


Figure 3.16: Cumulative distribution of the speedup of Hybrid with $\theta = 0.25$ when partitioning the benchmark problems into four (equally sized) groups based on which quartile in the distribution of the number of keys (left) or ranks (right) each problem falls into. On the left, the groups with greater number of keys (harder problems) observe greater speedups. On the right, groups with fewer ranks observer greater speedups.

3.7.4 Speedup Relative to Keys

We expect that as the TKR problems become larger (i.e., more keys), the speedups of our approach over the Sweep approach will increase. In the left plot of Figure 3.16, we show the distribution of the speedups of Hybrid with $\theta = 0.25$ on four equally sized groups of problems obtained from the TopKube-Benchmark. Again we ran each of these problems with $k = 5, 10, 20, 40, 80, 160, 320$. The first group has the first quarter of problems with the fewest keys, the second group has the next quarter of problems with next fewest keys, and so on up to the fourth quarter of problems with the most keys. From the plot, we observe the expected pattern: as problems become harder, the distribution is shifted to the right of the cumulative distribution plot, indicating larger speedup values. For example, on the intersection of the vertical line at $1\times$ (same speed), we already have more than 30% of the problems in group one (the easiest group), while this number is around 5% for the problems in group 4 (the hardest group). This is encouraging and supports using TopKube with Hybrid on even larger problems.

3.7.5 Speedup Relative to Ranks

We partitioned the problems in TopKube-Benchmark in four groups based on which quartile of the distribution of the number of ranks they fall into. In the right plot of Figure 3.16, we see an exact inversion of the pattern observed when we broke down the problems by the number of keys. The speedup is greater when fewer ranks are involved in a TKR problem. This is explained by the fact that more ranks yield larger values for i_{split} in the Hybrid Algorithm, and thus a longer sweep phase (Figure 3.6, Line 19).

3.8 Discussion

Potential Improvements. With the availability of the TopKube-Benchmark problems and the C++ reference implementation for the algorithms presented in this chapter, we would like to motivate new studies to improve on our results. One issue with Hybrid is its dependency on the parameter θ. Although $\theta = 0.25$ works very well, we see in Figure 3.14 that is not the fastest in every case. This suggests the need for an adaptive way to find i_{split} in Hybrid that is not based simply on a fixed θ.

The main focus of this work has always been computing top-k queries interactively; thus TopKube construction times and memory utilization were never optimized, but both can be greatly improved. The construction times (in hours) for the four datasets using a single CPU thread were respectively: 5.3h, 3.9h, 3.4h, and 1.7h. This yields insertion rates ranging from 4.7 to 6.5 thousand records per second. Preliminary experiments have shown that by using a multi-threaded build, these insertion rates can increase close to linearly with the number of threads. The memory (in gigabytes) used by TopKube for the datasets was respectively: 114GB, 20GB, 14GB, and 53GB. These numbers are significant, but again can be greatly reduced with an optimized implementation. We note that by utilizing path compression on sparse hierarchies (i.e., deep hierarchies with few branches) we can reduce the reported memory usage by more than an order of magnitude.

Conclusions. As user-generated online data continues to grow at incredible rates, ranking objects and information has never played such an important role in understanding our culture and the world. Although previous techniques have been able to create such rankings, they are inefficient and unable to be used effectively during an interactive exploration of the ranked data. We have introduced TopKube, an enhanced in-memory data cube that is able to generate ranked lists up to an order of magnitude faster than previous techniques. A careful evaluation of our techniques with public datasets has demonstrated its value. We have also made our benchmarks available for others to make direct comparisons to our work.

Chapter 4

City Geometry Data: Efficient Accumulation of City-Scale Shadows Over Time

A rapid increase in the urbanization of the world's population [3] has resulted in the need for cities to *densify* to equitably meet the rising housing demands while still maintaining the *environmental quality* of public spaces such as streets and parks. A key quantity that plays a crucial role in defining this quality is the impact of shadows from buildings. In particular, shadows can potentially infringe on the "right to light" of other citizens in the community through the occlusion of direct sunlight by shading public spaces. This can not only inhibit vegetation growth but also reduce solar energy potential. On the other hand, shadows can also be beneficial by reducing the urban heat island effect that paved surfaces create, or by providing a comfortable environment for park goers. It is therefore important to maintain a balance in the amount of shadows cast with the development of a city.

This requires extensive analysis to be performed to allow for amiable negotiations between the various stakeholders including the city council, the urban designers and developers, and other government agencies. However, in practice there is little analysis of cast shadows being done to test the impact of new development primarily due to the non-availability of the necessary tools. While cities do perform shadow analysis, the time and cost involved limits it mostly to a small and discrete set of times and very specific instances (e.g., see [106, 107]). It is therefore crucial that efficient and interactive tools are within purview of stakeholders since this allows for 1) a more comprehensive analysis; and 2) democratization of the planning process – making these results accessible and allowing the general public to visualize various scenarios will also help them contribute to the dialogue around new policies. Interactivity in such analysis also helps architects and developers quickly iterate over several possible designs when working on a project.

The computation of shadows is one of the most popular topics of research in the computer graphics domain. Due to its importance in realistic rendering, several techniques have been proposed for computing shadows both in real-time, as well as offline [108, 109]. These techniques were designed to support a variety of scenarios involving the scene as well as lighting options. All of these techniques, however, typically consider only scenes with a fixed set of light(s). Orthogonal to these techniques, we are interested in quantifying shadows over multiple time periods of interest. In particular, we are interested in quantifying the amount of time a given location is in shadow over the given time periods. This requires the *accumulation* of shadows involving different time periods of varying lengths. In addition, we are also interested in measuring how proposed developments can affect the accumulation of shadows in its neighborhood.

While none of the existing techniques directly support the accumulation of shadows over time, they can still be extended to accumulate shadows. The most straightforward approach, also followed by currently used tools [110, 111], is to explicitly identify shadows for each time step of a given interval. The direction of sun light depends not only on a city's location, but also on the time of the year. This makes the shape of shadow highly dependent on the day and time, requiring shadows to be computed for potentially several thousand time steps depending on the temporal resolution. Combining this with the scale of a city, which is typically spread over a wide area consisting of thousands of buildings, makes it computationally expensive to be performed at a suitably high resolution. More importantly, this increased complexity also hampers the *interactivity* of the analysis pipeline. For example, to accumulate shadows in a 3-hour period over a week at a time resolution of 1 minute would require shadows to be computed for 1260 light positions. Doing this for larger time periods spanning several months (such as summer or winter) at high resolutions is not practical.

Another option is to pre-compute all possible shadows, or appropriate data structures such as shadow maps or shadow volumes, and use this for the analyses. However, as mentioned above, we are interested in exploring the impact of new constructions with respect to the shadow. This requires interactively testing and iterating over several designs, and any pre-computation based approach will not be able to efficiently handle such scenarios, since the data structures will have to be recomputed based on the new set of conditions.

A third option is to model the given time interval as a set of directional lights. This approach requires the ability to support several thousand to tens of thousand light sources. However, existing techniques are catered towards only a small number of light sources, making such an extension non-trivial.

Contributions. In this chapter, motivated by problems faced by architects and city planners, we take the first step at interactively accumulating shadows over time. We first define two shadow accumulation quantities to effectively quantify the accumulation of shadows over time. We then propose a simple approach to efficiently accumulate shadows over time, and which can be used to speed up existing shadow techniques. Our shadow algorithms are then used to develop an interactive visual exploration system targeted at city planners and architects. Our contributions are summarized as follows:

- We propose a simple approach to accumulate shadows that implicitly tracks the movement of shadows. It is accomplished by taking advantage of the properties of sun movement within short time intervals.
- Our proposed approach is used to extend two common shadow techniques – shadow maps and ray tracing, to efficiently accumulate shadows. In particular, we present *shadow accrual maps* which extend standard shadow maps to accumulate shadows over time, and *inverse accrual maps* which use ray tracing to identify shadow movement, thus allowing shadows to be accumulated by simply drawing a set of lines.
- Making use of the coherency in the sun directions, we design optimizations that allow for the interactive accumulation of shadows over large time intervals, such as seasons.

- We show experimental evaluation demonstrating the accuracy and performance of our technique. We show that on average, the shadow accumulation using shadow accrual maps performs around an order of magnitude faster than a naive baseline.
- We develop *Shadow Profiler*, an interactive visual analysis system targeted at city planners and architects to explore shadows in a city, and test the impact of multiple scenarios.
- We show the utility of Shadow Profiler through case studies set in Manhattan, New York City. The case studies evaluate accumulated shadows over Central Park to study a set of *supertall* towers currently under construction that has generated intense public debate.

4.1 Related Work

We briefly survey existing literature from three categories: visual analytics in the context of cities, the study of shadows in urban design, and shadow computation techniques from computer graphics.

Urban visual Analytics. Multiple visual analytics systems have been proposed to interactively explore and analyze urban data [112]. These systems are primarily designed to analyze urban data generated by the urban environment. For example, there are individual visual analytics systems in transportation and mobility [23, 113, 114, 115, 116], air pollution [117], real-estate ownership [118, 119] and public utility service problems [120]. There are also tools developed to multiple urban data sets [22, 24]. We refer the reader to Zheng et al. [121] for a comprehensive survey on visual analytics approaches in urban computing.

Recently, several software platforms have also emerged that aim to use urban data sets together with city geometry to help inform the decision making process in the development of cites. They are aimed at a range of stakeholders (architects, city planners, developers, and the general public), such as Place I Live [122], Transitmix [123], Flux [124], ViziCities [125], ArcGIS [126], Urbane [24], and Vis-A-Ware [127]. Of these only Urbane, Vis-A-Ware and ArcGIS support computing impact based on measures such as visibility and sky exposure. While ArcGIS also supports the computation of shadows, it does not have the ability to accumulate shadows and visualize this accumulation.

Shadows in Urban Design. Sunlight exposure has been a core consideration in building design since early architectural studies. The seminal work of architect Ralph L. Knowles in which he proposed the concept of a *solar envelope* [128] has been hugely influential on studies involving the impact of shadows, and more generally solar access. This was further explored in the following decades, by Knowles [129, 130], and others [131, 132, 133, 134, 135], stressing the importance of solar access in the urban context.

In the specific context of shadows, Richens and Ratti [136, 137, 138] proposed a technique that computes shadow information based on digital elevation models (DEM) and used it as a parameter to generate and evaluate urban models. Shadows computed using DEM have also been incorporated into urban climate models [139, 140, 141, 142]. However, in all these cases, the computed shadow information is approximate since DEMs are not only limited by the resolution of the images, they also do not capture the actual shape of the buildings. Solar potential analysis also makes use of shadow information to improve the modeling of solar radiation, as well as assess photovoltaic energy-potential of urban environments [143]. These approaches also explicitly identify shadows for each time step of interest either through shadow maps [144] or using ray casting [145, 146]. Having an efficient approach to compute and accumulate shadows will greatly help in improving these models.

Shadow Computation Techniques. The computation of shadow information has been extensively explored in computer graphics. We refer the reader to two recently published books by Eisemann et al. [108] and Woo et al. [109] for a detailed survey on recent shadow computation techniques. Real-time shadow computation can be broadly divided into two categories – shadow map based techniques and shadow volume based techniques. The first group encodes shadow information of the scene geometry onto an image, which is later used when rendering the scene. Because the shadow information is discretized as an image, shadow map based approaches are constrained by the image resolution. Several solutions have been proposed to overcome this problem that include using multiple shadow maps [147, 148], pre-computing and storing high-resolution maps [149, 150], and deforming the light projection matrix in order to increase the texel density near the view camera [151, 152]. Sintorn et al. [153] proposed a shadow mapping technique that maintains a list of points (from the camera's POV) corresponding to each pixel of the shadow map to avoid the aliasing artifacts. Lokovic and Veach [154] stored multiple depth values as a parametric function to compute shadows for dense translucent objects such as hair and fur. The proposed shadow accrual map also uses an approach of storing multiple depth values, but the method for computing this is different since it has to consider different time steps. Scherzer et al. [155] presented a detailed survey on shadow map based techniques.

Shadow volumes based techniques, on the other hand, do not perform any discretization of the scene. Instead, it uses the geometry of the scene to create *volumes* of shadows in space. We refer the reader to Kolivand et al. [156] for a survey on shadow volume approaches. Recently, Sintorn et al. [157] proposed a shadow volume technique that assigns a volume for each triangle in the scene. However, since this requires pre-computation, impact computation in real-time becomes difficult. As mentioned in Section 1, accumulating shadows using any of the above techniques requires explicitly computing shadows at each time step, making it an expensive process.

With the current progress of massively parallel architectures, another approach that has become popular for computing shadows is ray tracing. Djeu et al. [158] used volumetric occluders to accelerate the tracing of rays for shadows. Kalojanov et al. [159] stored the scene using a two-level grid to enable interactive ray tracing using GPUs. Feltman et al. [160] proposed a cost estimator for shadow ray traversal, and used it to indicate early ray termination. Nah et al. [161] proposed a surface method traversal order that accelerates shadow ray tracing.

Soft shadowing techniques [153, 162, 163, 164] can be used to obtain the desired visual effect of shadow accumulation, by considering samples to correspond to the time steps. This still boils down to explicitly computing shadows at each of the time steps. Also, using all of these techniques, it is not possible to quantify the shadow contribution with respect to the source, which is important for analysis.

To the best of our knowledge, the only approach that computes shadows over time of day was proposed by Fernando [163, Chapter 13], which pre-computes *occlusion interval maps* that store for each point the time steps when they are visible to light. This is a costly pre-computation, which cannot be easily adjusted to interactively compute impact with changes to the scene. Orthogonal to these techniques, our approach uses the property of shadow movement over time to accumulate shadows in real-time. Moreover, our approach can be used to extend any of the above techniques to speedup shadow accumulation.

4.2 Temporal Shadows

In this section we first formally define two measures to quantify shadow accumulation followed by describing the key property of temporal shadows, that form the basis of our shadow accumulation techniques.

4.2.1 Shadow Accumulation

A given location can be in shadow at different times of a given day. Our goal is to measure the quantity of shadow at these locations. In particular, we are interested in the following quantities which define two different aspects of a shadow with respect to a location:

Gross Shadow. measures the total time that a given location is in shadow during a given time interval. When computed over multiple days, we compute the gross shadow as the average time per day that location is in shadow. For example, consider the shadows with respect to two towers in Figure 4.1 for a 3-minute interval[1]. Point p_1 is in shadow for the entire time interval, while points p_2 and p_3 are in shadow for 2 minutes of the interval.

Continuous Shadow. measures the maximum time that a given location is continuously in shadow during a given time interval. When computed over multiple days, it is equal to the maximum continuous duration over all days. Again, consider the example in Figure 4.1. Points p_1 and p_2 are continuously in shadow for 3 and 2 minutes respectively, which is the same as the total time (gross shadow) they are in shadow. On the other hand, even though p_3 is in shadow for 2 minutes, by having no shadow at 12:01 PM, it is continuously in shadow only over 1-minute intervals.

[1]For illustrative purposes, the shadow between consecutive minutes are exaggerated. In reality, the shadows are much closer.

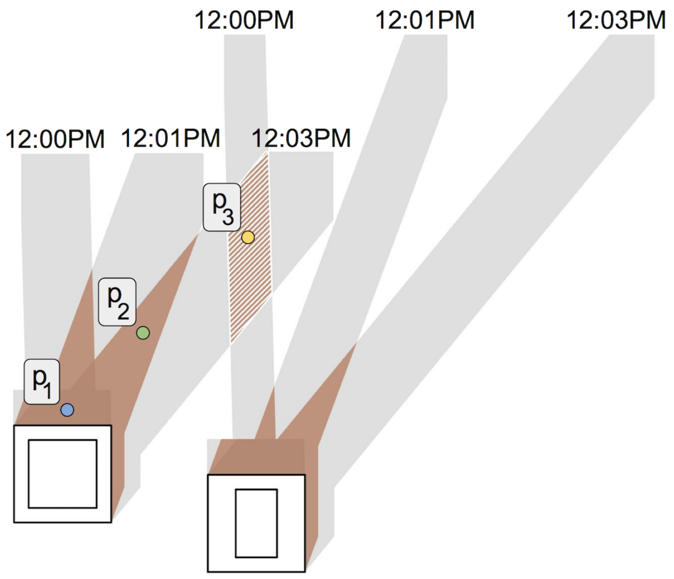


Figure 4.1: Shadow accumulation is measured as either gross shadow or continuous shadow. In the above example, the point p_2 has both measures equal to 2 minutes in the 3-minute interval. The point p_3, on the other hand, is continuously in shadow for only a minute, even though it is in shadow for 2 minutes in this interval.

4.2.2 Properties of Temporal Shadows

One way to accumulate shadows in a given time interval is to compute shadows at each time step of this interval and combine them. However, as mentioned earlier, this is a costly operation and is not interactively feasible even when the accumulation is done only over a single week. Rather than tracking the movement of the sun (directional light source) over time, the key idea behind our technique is to alternatively track the movement of the shadow itself in order to accumulate them. For the remainder of the chapter we assume the light source as directional.

Consider the time interval $[t_1, t_n]$. Given a relatively short time interval, the movement of the sun during this interval can be considered to be linear. To validate this assumption in practice and to identify an appropriate time interval, we compare the actual sun direction with the interpolated direction over different interval sizes. In particular, we first choose 1000 random time steps covering the entire year. This ensures that directions from different times of the day as well as different seasons are well covered. Given a time interval of n minutes, we compute the

Table 4.1: Mean (μ), standard deviation (σ) , and median of the cosine value between the actual direction of sun light (in NYC) and the direction obtained by linearly interpolating between different time interval sizes. Note that the linear approximation starts diverging from the actual direction only when the interval size is greater than an hour.

Minutes	5	10	30	60	120	240
μ ($\times 10^{-1}$)	9.9999	9.9999	9.9999	9.9999	9.9997	9.9952
σ ($\times 10^{-6}$)	1.31	0.93	1.90	4.92	25.5	415
Median	1	1	1	0.99999	0.99998	0.99965

cosine between the actual sun direction at each minute and the direction obtained by interpolating between the directions at the start and end of that interval. A value close to 1 indicates that the two direction vectors are the same. Table 4.1 shows the mean, standard deviation, and median of the cosine values with different interval sizes ranging from 5 minutes to 4 hours. Note that the linear assumption of the sun movement holds even when the value of $n = 60$ minutes. We start seeing the interpolated directions diverging from the actual direction beyond this interval. We therefore decided to use *hourly intervals* ($n = 60$ minutes) to compute shadow accumulation.

The main idea behind our approach is the following. Consider point p_1 on the ground that is in shadow at time t_1. As illustrated in Figure 4.2, let the cause of shadow at p_1 be the point s on a building. Note that s can be one of the many possible sources of shadow at p_1. Let at time t_n, s cast a shadow at point p_n. Then,

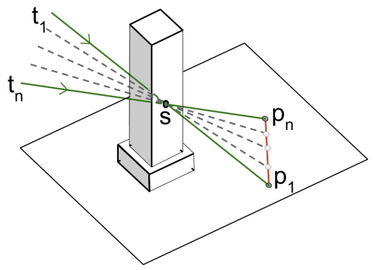


Figure 4.2: Shadow accrual map makes use of the linear movement of the sun over short time periods to track the movement of shadows. Given a time interval, $[t_1, t_n]$, each point p_1 in shadow at t_1 is mapped to the point p_n, the location of shadow at t_n due to the same shadow source, s.

given that the movement of the sun is linear, the shadow cast by s moves linearly from p_1 to p_n. Thus, the accumulated shadow corresponding to s over the given time interval is essentially the straight line from p_1 to p_n. This key observation is used in the next two sections to design algorithms to efficiently compute shadow accumulation over time.

4.3 Shadow Accrual Maps

We now describe *shadow accrual maps*, an extension to the standard shadow mapping technique [165] that utilize the linear shadow observation to keep track of shadows over time. The standard shadow mapping algorithm runs in two passes. First, it renders the scene from the point of view of the light (using an orthographic projection in the case of a directional light), and stores the depth buffer in a texture called the shadow map. The shadow map maintains the distances between the light and the objects that are directly illuminated. In case of directional light, the distance is computed with respect to a plane orthogonal to the light direction. Next, the scene is rendered from the point of view of the camera. The depth of the surface point corresponding to each pixel is computed from the light's point of view as above. If this depth is greater than the depth stored in the corresponding pixel in the shadow map, the point is marked as being in a shadow.

Our algorithm follows the same template, wherein the first step computes the shadow accrual map for a given fixed time range, and the second step identifies points in shadow.

Step 1: Computing Shadow Accrual Maps. Consider a given time range $[t_1, t_n)$ in which the movement of the sun is linear. Let this time range be divided into n discrete time steps. The shadow accrual map is a 3D texture that stores the depth values corresponding to these n time steps. However, instead of individually computing the n 2D textures (or shadow maps) over n passes, we compute it in one pass as follows.

Let $\vec{d_1}$ and $\vec{d_n}$ be the direction of sun light at the beginning and end of the given time range. We select a shadow plane that is orthogonal to $\vec{d_1}$, and further from the light than all objects visible from the camera as shown in Figure 4.3a. The extent of this plane is computed such that it encompasses all objects from the point of view of the camera when projected with respect to directions $\vec{d_1}$ and $\vec{d_n}$.

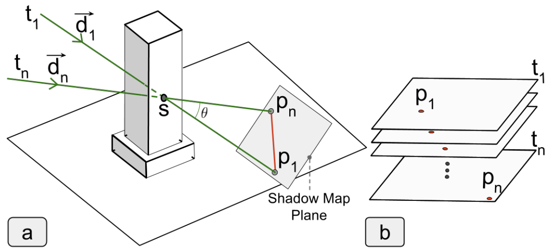


Figure 4.3: Shadow accrual map is a 3D texture, where each slice stores the depth values corresponding to a single time step. The depth value for a given time is assigned by interpolating the shadow from its projection at time t_1 to its projection at time t_2.

Every 3D point s in the scene is processed as follows. s is first projected onto the shadow plane along directions $\vec{d_1}$ and $\vec{d_n}$ to obtain points p_1 and p_n respectively. That is, p_1 and p_n correspond to the locations of the shadow cast by s at times t_1 and t_n respectively (see Figure 4.3a). Since the shadow moves linearly within the given time interval, the location p_i of the shadow at every intermediate time step t_i, $1 < i < n$, is computed as $p_i = p_1 + (p_n - p_1) \times \frac{\tan(i'\theta/n)}{\tan(\theta)}$, where $\theta = \angle(\vec{d_1}, \vec{d_n})$, and $i' = i - 1$. For each i, shadow depth of p_i in the i^{th} 2D slice is appropriately updated as shown in Figure 4.3b. Instead of the distance between s and the light, we use an equivalent measure of the distance of s to the shadow plane along the light direction as depth. Thus, the depth of p_i is simply the distance between s and p_i.

Using modern programmable GPUs, the entire operation can be performed in parallel in a single rendering pass. As the following theorem shows, the resulting 3D texture is equivalent to creating independent shadow maps for each of the n time slices. Thus, there is no loss of quality when using shadow accrual maps compared to traditional shadow maps.

Theorem 1. Consider a time interval of size n units during which the movement of the sun is linear. Let shadows be accumulated for every 1 unit of time, i.e., the time interval is divided into n equal time steps. Then the shadow accrual map computed for this time interval is the same as the computing n shadow maps for each of the n time steps.

Proof. Consider a time interal $[t_1, t_n]$ of length n units. Without loss of generality, let $t_1 = 0$ and $t_n = n - 1$. Let $\vec{d_1}$ and $\vec{d_n}$ be the direction of light at t_1 and t_n

respectively. By definition, the first and last slices (corresponding to time t_1 and t_n) of the shadow accrual map are the same as the shadow maps for these two time steps.

Now, consider time step $t_1 < i < t_n$, having direction $\vec{d}$. Due to linear movement of the sun, the interpolation factor $k = \frac{i-t_1}{t_n-t_1} = \frac{\angle(\vec{d},\vec{d_1})}{\angle(\vec{d_n},\vec{d_1})}$. Let $\angle(\vec{d_n},\vec{d_1}) = \theta$. Then $\angle(\vec{d},\vec{d_1}) = i\theta/n$. Consider a point s on building mesh. Let the projection of s on the shadow plane be on point p at time i. Without loss of generality, we use p to denote the pixel on the shadow map texture as well. Therefore, the point p will be processed by the traditional shadow map algorithm for time step i. Let the shadows at times t_1 and t_n be at points p_1 and p_n respectively. Note that the direction $\vec{d_1}$ is orthogonal to the shadow plane. Therefore, the point p is at a distance $\delta \tan(i\theta/n)$ from p_1 along the line $[p_1, p_n]$ (see Figure 4.4).

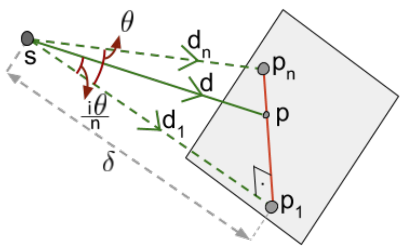


Figure 4.4: The pixel p processed by the shadow map at time i is obtained by projecting along the light direction $\vec{d}$, which is at an angle $i\theta$ from $\vec{d_1}$.

Now consider the shadow accrual map algorithm. At time i, it processes the point $p' = p_1 + (p_n - p_1) \times \frac{\tan(i\theta/n)}{\tan(\theta)}$, which is the same as p, for the i^{th} slice. Thus, for any shadow source s, shadow accrual map processes the exact same pixels for all time steps i as a shadow map would for the corresponding directions. □

Step 2: Computing Shadows. To obtain the shadow at a given 3D point s, we need to test its depth at the n time steps. To do this, consider again the line between the projection of s at time t_1 and time t_n. As before, let p_i be the projection of s at time steps $1 < i < n$. If s is in shadow at time step i, then the depth of p_i will be less than the corresponding depth stored in the i^{th} 2D slice of the shadow accrual map (recall that the depth is the distance between s and p_i). The gross shadow is then computed by simply counting the number of points p_i, $1 \leq i \leq n$, that are in shadow. The continuous shadow is computed by counting the maximum number of points that are continuously in shadow. Note that if each time step is different from 1 minute, then the gross (continuous) shadow is multiplied by an appropriate factor.

4.4 Inverse Accrual Maps

The primary application of accumulated shadows in the context of cities is in studying its impact on open urban spaces (such as parks or sidewalks), which are typically flat surfaces. Since shadow accrual maps are based on shadow maps, the well known issues such as aliasing and shadow acne are also carried over making accurate quantification of shadows difficult. Ray tracing based shadow techniques, on the other hand, have better quality. With the focus on shadow accumulation over flat surfaces, in this section, we design a ray tracing-based approach that makes use of the linear movement property of temporal shadows.

Inverse Accrual Maps. Consider again the example in Figure 4.2. Given that the accumulated shadow corresponding to s is the straight line from p_1 to p_n, the inverse accrual map maps the point p_1 to the point p_n. It is computed as follows. First, the set of points on the plane visible from the camera are identified. This can be accomplished by a simple modification of the rendering output of the graphics pipeline where the world coordinates of each pixel is stored onto a buffer.

Now, consider any point, say p_1, on a plane. The possible sources of shadow for that point can be obtained by tracing a ray from that point in the reverse direction of light until there are no more intersections. Here, each intersection corresponds to a source of shadow. The inverse accrual map is also a 3D texture, where the i^{th} 2D slice stores the mapping corresponding to the i^{th} source. Figure 4.5 shows two points for which there are two source points. If there is no shadow on p_1 at t_1, then such a point has no source of shadow, and is hence mapped to infinity. To avoid shadow acne, we ensure that there is a small distance between p_1 and its shadow source (we notice that in practice, a distance of 100 cm provides good results).

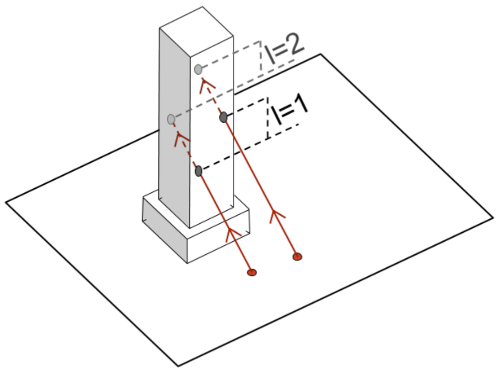


Figure 4.5: A given view point can have more than one source of shadow. The l^{th} closest source of shadow is used to represent the l^{th} 2D slice of the inverse accrual map.

Computing Shadow Accumulation. Let the given time interval be divided into n time steps. The shadows corresponding to each of the source levels are first drawn as follows. Consider a point p_1 and the corresponding mapping point p_n. The shadow at each time step is approximated to be along one of the line segments obtained by dividing the line (p_1, p_n) into $n-1$ segments. We maintain a n-bit vector for each point to store the shadow corresponding to it at the different time steps. Consider a point p. The bit corresponding to the j^{th} time step is set to 1, if the j^{th} line segment pass through this point. For example, consider the illustrated points in Figure 4.1. Given the 3-minute time interval with $n = 3$, each point has a 3-bit vector associated with it. The vector corresponding to p_1, p_2, and p_3 are $[1,1,1]$, $[0,1,1]$, and $[1,0,1]$ respectively.

After all n-bit vectors corresponding to points in the scene are populated by drawing all valid lines from all source levels, gross shadow is computed as the sum of bits in this vector. Continuous shadow is computed as the maximum size of a set of consecutive 1's in the vector. As before, gross (continuous) shadow is multiplied by an appropriate factor to offset the size of a time step.

Effect of Maximum Source Level. Consider the evolution of the shadow at point p in Figure 4.6 from time t_1 to time t_n. Let the line (p_1, p_n) be responsible for the shadow at p at time t, where $t_1 < t < t_n$. Let s be the source of this shadow. For the point p to be correctly identified as being in shadow at t for source level $l = 1$, the corresponding inverse accrual map should associate point p_1 to p_n. However, as shown in Figure 4.6, this is not true because source level $l = 1$ uses the closest shadow source, which in this case is not s.

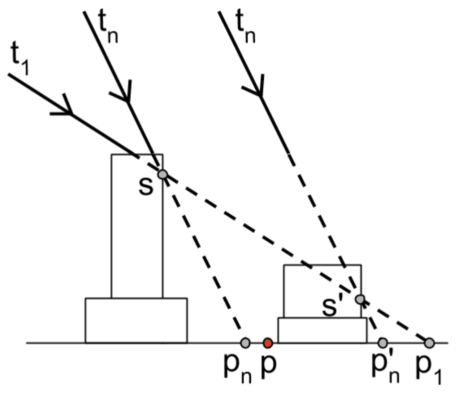


Figure 4.6: Possible situation which requires inverse accrual maps to be computed at multiple source levels.

In order to obtain accurate shadow accumulation, it is therefore necessary to compute inverse accrual maps over all possible source levels. This becomes expensive especially during dawn or dusk, since the light direction from the sun is close to horizontal and a ray in the reverse light direction can intersect several buildings. However, in such a scenario, the movement of shadows due to farther away points is very fast, causing little loss of accuracy if these points are omitted. Thus, to maintain practical computation times, we can limit the maximum value of l when computing the inverse accrual map. Also, when accumulating shadows from time t_1 to t_n, in addition to computing inverse accrual maps from t_1 to t_n, we also compute them from t_n to t_1. This serves three functions: (1) during the later part of the day, the shadow stretches with time. Thus, a point in shadow at t_1 would correspond to an area of points at t_n. However, the inverse accrual map associates only a single point at t_n, and the drawn shadow line will not reflect this stretch. By reversing the time interval, the map would then correspond to contracting shadows and will ensure that no points are missed during shadow accumulation; (2) in the example from Figure 4.6, note that p_n for the time interval (t_n,t_1) is mapped to p_1. Thus, the point p is not problematic anymore when $l = 1$. This also helps improve accuracy while still maintaining a small number of 2D slices; and (3) when the ray at time t_1 is parallel to a building facade, these instances are captured in inverse accrual maps from t_n to t_1, thus improving the accuracy of the approach.

For the remaining of the chapter, when using inverse accrual maps, we compute the map along both t_1 to t_n and t_n to t_1. With this addition, as we show later in Section 4.7.2, the accumulated shadow converges close to its true value with very low error when the source level $l \leq 3$.

Discussion. Given that the computation of inverse accrual maps identifies the sources of shadow, a simple modification to keep track of this will allow the identification of the source of the shadow – the object(s) causing the shadow. As we show later, this is useful for analyzing shadows and their causes in cities.

4.5 Handling Large Time Intervals

The shadow accumulation using either of the above two approaches is computed for short time intervals (60 minutes) when the movement of the sun can be approximated to be linear. Therefore, when required to accumulate shadows spanning multiple days (or months), one way to accomplish this is to explicitly compute shadow accrual maps for all 60-minute intervals at a resolution of 1 minute (i.e., $n = 60$) corresponding to the given time period.

While the direction of sun light at a given time in summer will be drastically different from the direction in winter at the same time (depending on the geographical location), the change in direction on consecutive days in summer (or winter) is minimal. We use this key observation to significantly reduce the number of shadow accrual maps (or inverse accrual maps) that are computed, as shown next.

For a city of interest, in a preprocessing step, we first cluster all possible light directions into a set of bins. Consider a ray along each light direction originating from a reference point, which is the origin. Then, the bins are defined by partitioning a hemisphere, that is centered at this origin, into quads such that the maximum angle (azimuthal and polar angle) corresponding to any quad is bounded. Using a sufficiently small bound, any light direction can be represented by the bin it is associated with.

Now, let the shadow be accumulated from time t_{start} to t_{end} for a period of d days. So each day will require shadow accrual maps (or inverse accrual maps) to be computed for $k = (t_{end} - t_{start})/n$ time intervals per day. This can be represented as a collection of pairs $(\vec{s_j}, \vec{e_j})$, $1 \leq j \leq k$, where $\vec{s_j}$ is the start light direction and $\vec{e_j}$ is the end light direction for the j^{th} 60-minute interval. We create a weighted graph $G(V,E)$, called *direction graph*, where each node in V corresponds to one bin of the above described index. There is an edge between two nodes if there exists a direction pair $(\vec{s_j}, \vec{e_j})$ corresponding to those bins. The weight of an edge is the number of times that pair is present for the given time interval. For example, consider a case where shadows have to be accumulated over two days from 10 AM to 3 PM. Let the value of $n = 60$ minutes. If the direction of sun light remains the same for both days until 1 PM, then the resulting graph is as shown in Figure 4.7. For the first 3 hourly intervals, the edge weights will be 2 since the corresponding directions are common between the two days.

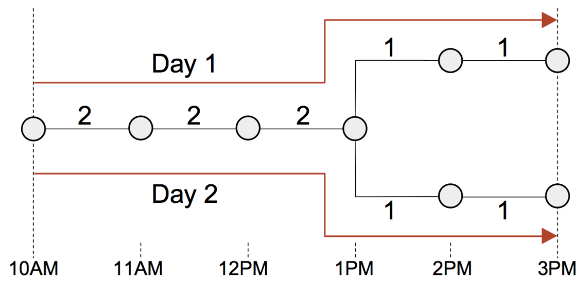


Figure 4.7: The direction graph is used to significantly improve the performance of shadow accumulation.

Accrual maps now have to be computed only once for each edge. This number is significantly smaller than explicitly computing them for all $k \times d$ intervals (see Section 4.7.2 for more details). Thus, in the above example, shadow accrual maps (or inverse accrual maps) have to be computed only for 7 one hour intervals instead of 10 one hour intervals (i.e., $k = 5$ hourly intervals over $d = 2$ days).

Given this setup, the different shadow accumulation quantities are computed as follows.

Gross Shadows. A given n-minute interval corresponds to an edge in the direction graph. Let G_j be the gross shadow computed for edge j in the graph. When considering multiple such intervals, the gross shadow is equal to the sum of gross shadows computed from each interval. Given the direction graph, this sum is equal to:

$$G = \sum_{j=1}^{k} G_j \times w_j \tag{4.1}$$

where w_j is the weight of the corresponding edge.

Continuous Shadows. Consider an edge in the direction graph and the associated shadow accrual maps. When computing continuous shadows for each point in the given interval, in addition to the maximum continuous shadows for the corresponding interval, we also store the length of the longest prefix and longest suffix of continuous shadows (these will be the longest prefix and suffix of 1's from the n-bit vector in case of using inverse accrual maps). The movement of sun on each day corresponds to a path of edges in the direction graph. For the example in Figure 4.7, paths corresponding to the two days is illustrated in red. To compute the continuous shadow over these edges, the corresponding accrual maps are processed in the order of the path traversed. In particular, the prefix, maximum and suffix values are used to "stitch" together consecutive edges. Note that these values are computed only once for each edge, and reused multiple times. To avoid the number of accrual maps that are cached in memory, we traverse the paths in a topological order so that accrual maps can be discarded as soon as all paths using them are processed.

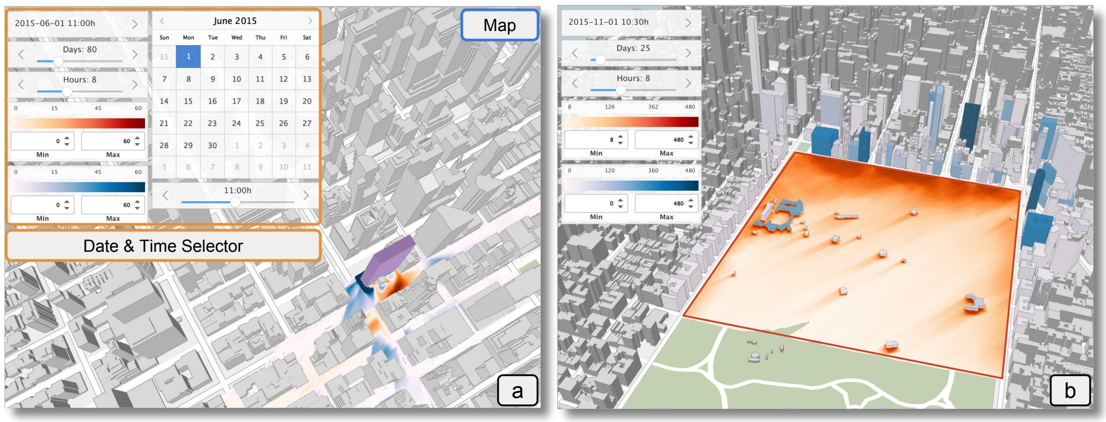


Figure 4.8: User interface of the Shadow Profiler system consists of a map widget together with a date and time selector. **(a)** We analyze the shadow impact when inserting a new building. The divergent color map highlights areas where the new building would add shadows (in red), or areas where it would decrease shadows (in blue). The scale of the color map is in minutes, and can be adjusted by the user. **(b)** Visualizing the shadow contribution of buildings with respect to accumulated shadows in the selected region; a darker shade of blue indicates a higher contribution by that building. The accumulated shadows are visualized using the color map shown in the interface.

4.6 Shadow Profiler

The shadow accumulation approaches are used to design *Shadow Profiler*, a visual exploration system that allows users to explore and analyze shadows in a city. We now briefly describe its visual interface and discuss the analysis measures it supports.

4.6.1 Visualization Interface

The interface (Figure 4.8) is primarily composed of two components: 1. a *3D map* widget that provides spatial context; and 2. a *date & time selector* widget that allows for the user to select a time period of interest. A time period is selected by specifying four values – a start date D_{start}, start time t_{start}, number of days $n \geq 1$, and hours per day $k \geq 0$. A value of $k = 0$ represents a single time instant, and the shadow corresponding to the selected date and time is visualized. When $k > 0$, shadows are accumulated for k hours per day – from t_{start} to $t_{start} + k$ over a period n days starting from D_{start}. The accumulation type, which is one of

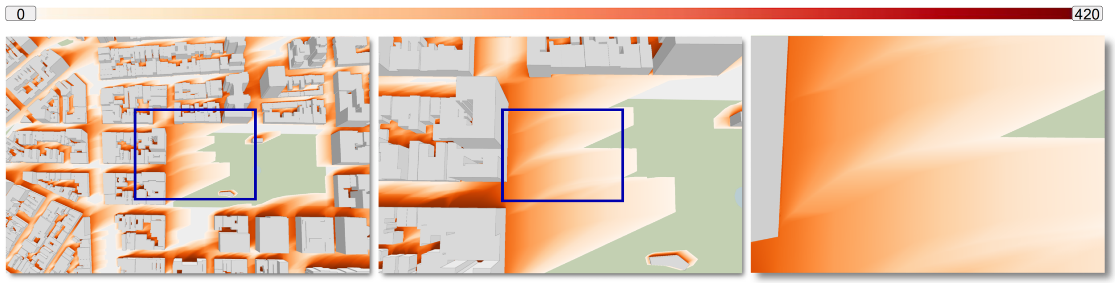


Figure 4.9: Shadow accumulation over 7 hours on June 1 at three different zoom levels when the camera is 800, 300 and 100 meters, respectively, above the ground. Note that the level of detail, as well as the quality of the visualization improves as the user focuses into a region of interest.

gross or continuous shadow, is selected by the user. The accumulated shadow is averaged over the number of days, and is visualized using a color map (Figure 4.8b). User interactions (pan, zoom, etc.) recompute shadows on the fly for the region corresponding to the viewport, thus enabling a level-of-detail rendering. Figure 4.9 visualizes the shadow accumulation over a single day at three different zoom levels focusing on Washington Square Park in New York City. Accumulating shadows over a large number of days can still take few seconds depending on the approach used (see Section 4.7.2). To support seamless interaction, we allow for a progressive computation and rendering of the shadow accumulation. We also allow users to brush and select polygonal regions of interest to inspect shadows. In this case, the visualization is restricted to the specified polygon.

An important task is the assessment of shadow impact with respect to a new building. To support this, we allow the user to select either an empty building lot, or an existing building that is to be demolished, and replace it with a user generated mesh. The shadows are then updated to reflect this change; this is accomplished by computing the difference of shadows between the two states and visualizing the result using a divergent color map. Figure 4.8a illustrates this task.

We support two visualization modes corresponding to the two approaches, as shown in Figure 4.10. Users can choose the mode based on their objective – the *exploration mode* is used for interactive visualization when users are interested in exploring the city, and uses shadow accrual maps; and the *analysis mode* is used when users are interested in a more detailed analysis and for computing the different analysis measures (described in the next section), and uses inverse accrual maps.

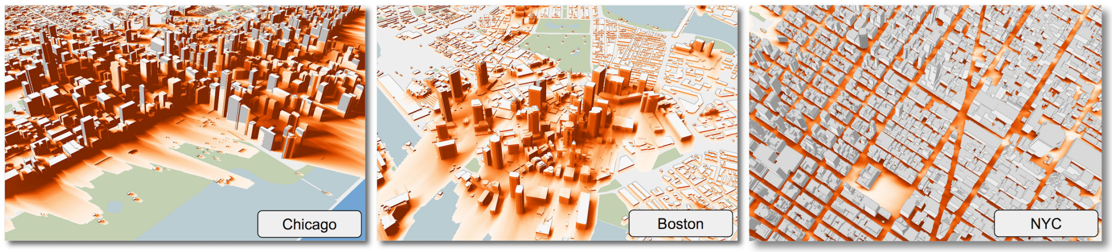


Figure 4.10: Shadow profiler supports two modes of operation. The exploration mode, which uses shadow accrual maps to compute shadow accumulation, is used to explore Chicago and Boston. The analysis mode, which uses inverse accrual maps, is used for New York City. The shadows were accumulated for 7 hours on March 28.

4.6.2 Analysis Measures

In addition to visualizing the accumulation, we also compute three different metrics quantifying the properties of the shadow. All of these quantities are computed with respect to a polygonal region R of interest selected by the user.

Shadow Area. Let $p \in R$ be a point that is within the selected region. The shadow area is computed as $Area = \int_{p \in R} shadow(p)$ where $shadow(p)$ is defined as follows. When a single time instant is being visualized, then $shadow(p) \in \{0, 1\}$ indicating the absence / presence of a shadow. When accumulating, $shadow(p)$ indicates the fraction of the time (gross or continuous) per day that point is in shadow. In a discrete setting, this value is equal to $\sum_{p \in R} shadow(p) \times area(p)$, where p represents a pixel, and $area(p)$ is the area covered by the pixel in square meters. To maintain accuracy, shadows rendered at a high resolution are used for this computation.

Shadow Score. As mentioned earlier the effect of shadows can be both positive as well as negative. For example, from a pedestrian point of view, shadows are preferred during summer since it makes the environment more comfortable, while disliked in winter. To evaluate this effect, we define the shadow score as:

$$Score = \int_{p \in R} \sum_{t \in T} (\omega_t \times shadow_t(p))$$

Here, the user divides the selected time period T into a set of time intervals t, and assigns a weight -1 $\leq \omega_t \leq +1$ for each interval, indicating the nature of shadow for that interval. For example, the user could assign a weight -1 for winter months, +1 for summer months, and 0 for other months. $shadow_t(p)$ specifies the fraction of the time per day a given point is in shadow during the interval t. In addition to computing the score, it is also visualized using a divergent color map (see Figure 4.16).

Building Contribution. The framework also allows for evaluating the shadow contribution of buildings over the selected region r. Here, each building is assigned a quantity equal to the shadow area resulting from that building, and visualized using a color map as shown in Figure 4.8b.

4.7 Implementation and Experiments

The shadow accumulation techniques were implemented using C++, OpenGL 4.3, and OpenCL 1.2. We now describe the implementation choices made, and discuss results from our experiments evaluating the performance of the two approaches.

4.7.1 Implementation

Shadow Accrual Maps. As mentioned earlier, the problems that are common with shadow maps also carry over to shadow accrual maps. To obtain better quality shadows at a lower resolution itself, we chose to use the trapezoidal transformation [166], which warps the shadow depth texture onto a trapezoidal approximation of the view frustum. This however doesn't solve the problem of shadow acne. So we use a bias offset to reduce the effect of shadow acne.

When computing the shadow accrual maps, since OpenGL does not allow more than 8 frame buffer objects (and thus depth buffers), we use the features of OpenGL 4.3 to store the depth values onto a 3D image texture. By making use of atomic operations on images, we are able to store, in a single rendering pass, the largest depth value of a texel, for all slices of the shadow accrual map.

Inverse Accrual Maps. As mentioned in Section 4.4, computing inverse accrual maps is accomplished by tracing a ray along the reverse light direction. Given a maximum source level (see Section 4.7.2), the ray is traced until either the given number of intersections is reached, or no other intersections are possible.

Our implementation uses a 3D grid to index the model of the city to be used for ray tracing. The corresponding accrual map associations are simultaneously computed during the ray traversal to output the inverse accrual maps. This part was implemented using OpenGL shaders. Note that the ray traversal also takes into account new buildings that are added or replaced.

OpenCL is then used to accumulate shadows – i.e., draw the shadow lines and perform the appropriate bit operations based on accumulation type. When rendering large time intervals, the computed values are combined with the existing values to enable progressive rendering. User specified operations such as computing analysis measures and impact are also performed at this stage.

Baseline Implementations. We implemented two baselines, based on shadow maps and ray tracing, to evaluate the performance of the proposed approaches. The shadow map baseline explicitly computes shadow maps for every minute, and uses them to identify and accumulate shadows. To maintain a consistent quality, we use the trapezoidal transformation for this implementation as well.

The brute force ray tracing based approach explicitly identifies the shadow for every minute in the time interval by tracing a ray from all pixels, which are then accumulated together.

4.7.2 Experiments

In this section, we first discuss results from our experiments evaluating the different parameters affecting the accuracy-time trade-off. We then report the performance of our technique when using the identified parameter values. The experiments were performed on a workstation with a Intel Xeon E5-2620 CPU, 128 GB RAM, and an Nvidia GTX 1080 graphics card with 8 GB RAM. We use Manhattan as the test bed for the experiments. The geometries of the buildings in the city were obtained through Open Street Maps and consist of over 43 thousand buildings present in Manhattan. The mesh is composed of 1.5 million triangles.

Accuracy Trade-Off Due to Direction Graph. A crucial step in improving the performance is grouping the set of possible light directions into a set of clusters, and using a representative of each cluster to approximate the light directions (Section 4.5). As mentioned earlier, the maximum angle between any two directions in a cluster is bounded by a specified angle. A small angle, while having high

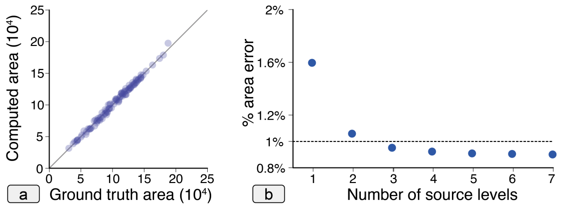


Figure 4.11: **(a)** Comparing the area of accumulated shadows over 1 hour periods computed using a smaller set of representative directions from the direction graph with ground truth area. Note that approximating using the direction graph does not hamper the accuracy of the shadow area. **(b)** Choosing the maximum source level for inverse accrual maps.

accuracy will impede the efficiency. On the other hand, a large angle can drastically decrease the accuracy. We found that using a bound of $5°$, we were able to get a good accuracy-time trade-off. In particular, when testing $n = 1000$ random time steps, and computing the similarity between the actual direction, and the direction of the cluster representative, we found that the mean similarity measure was 0.9996 with a standard deviation of 3×10^{-4}, implying that the clustering provides good approximation.

To quantify the effect this approximation has on the accumulated shadows, we chose a set of random camera positions and 1 hour time intervals, and computed the gross shadow when using both the actual direction (ground truth) and the cluster representative. Note that the gross shadow for this experiment was computed using the ray tracing baseline. The shadows were computed at a resolution of 800×600.

Figure 4.11a plots the ground truth shadow area against the shadow area computed using the cluster representative. Recall that the shadow area is the weighted sum of pixel area, weighted by the gross shadow. The mean and median absolute error in the area was only 0.47% and 0.35% respectively, with a standard deviation of 0.37%, when compared to the total area. On an average, only 0.8% of the points, with a standard deviation of 0.5%, were incorrectly tagged as being in or not in shadow.

Maximum Source Level for Accurate Shadow Accumulation. The primary use of inverse accrual maps is to accurately estimate the shadows for analysis. An important parameter affecting the accuracy of this approach is the maximum source level, i, that specifies the number of 2D slices of the inverse accrual map that is

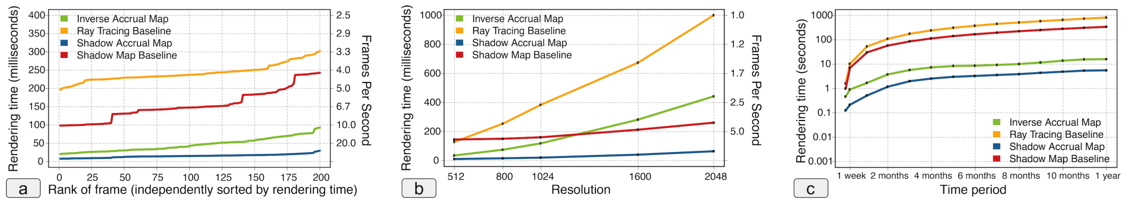


Figure 4.12: Performance Evaluation. **(a)** Comparison with baselines. For each method, the times are independently sorted in increasing order. Note that both shadow accrual map and inverse accrual map consistently perform better than the naive baselines. **(b)** Scalability of the proposed techniques with increasing resolution. We used resolutions with an aspect ratio of 1:1 for this experiment (e.g., 512 implies a resolution of 512×512). **(c)** Scalability with increasing time periods. Note the significant speedup (over 50X) achieved with increasing time periods (y-axis is in log scale).

to be computed. Recall that, given a time range $[t_1, t_n]$, the map is computed for both t_1 to t_n, as well as t_n to t_1. To identify a suitable value, we chose a set of random hourly intervals and camera positions, and compared the computed gross shadow between the ray tracing baseline (ground truth) and inverse accrual maps by varying the maximum source level.

Figure 4.11b plots the average percentage error in the shadow area with increasing source levels. As expected, the error decreases with increasing number of levels. Using this plot, we fix the knee of this curve, i.e., $i = 3$, as the parameter for computing inverse accrual maps. At this point, the mean error is less than 1% of the total area. The median error for $i = 3$ is 0.7%, while the maximum error is 2.6%. We found that the maximum error occurred primarily when the accumulation was performed during dawn or dusk. This is because the shadows are not only long, but they also move quickly. In such a case it is possible for a single point to have several sources of shadow. Since we are considering only 3 sources, we miss considering shadows that are due to other sources.

Performance Evaluation. For the remainder of this chapter, we use the parameters identified in the above experiments for computing shadow accrual maps as well as inverse accrual maps. We compare the effect of these parameters with both the baselines. These experiments consider the end-to-end time, which includes computing shadow accrual maps (or inverse accrual maps), and using them to compute and visualize the shadow accumulation.

For the first experiment, we consider 10 random positions, and 20 random days spread throughout the year. For each position-day pair, we compute the gross shadow for a period of 6 hours, starting from 9 am till 3 pm. While the selected days cover the different seasons of the year, using a period of 6 hrs ensures that the different positions of the sun during the day are considered as well. All shadows for this experiment were accumulated at a resolution of 800×800. Note that this was the output resolution. The shadow map and shadow accrual maps were at a resolution of 1024×1024. Since inverse accrual maps compute shadow accumulation only along the ground plane, we modified the ray tracing baseline to also do the same. Figure 4.12a plots the average time taken to compute shadow accumulation for an hour over the different days and camera positions. The reported time corresponds to the median computation time over 5 independent runs. On average, shadow accrual maps perform over 10X faster than the shadow map-based baseline, while inverse accrual maps perform around 5.3X faster than the ray tracing baseline.

The second experiment tests the scalability of the approaches. We fixed a position and day, and computed the gross shadows for same period of 6 hours, but varying the resolution. For this experiment, we set the output resolution the same as the shadow map resolution. Figure 4.12b plots the average time taken to render an hourly interval with increasing resolution. Note that both shadow accumulation approaches scale linearly with resolution.

Performance Improvement Due to Direction Graph. When accumulating time periods involving multiple days, a brute force approach would explicitly compute shadows for every minute over all days. On the other hand, when using the direction graph there is a reuse of the shadow accrual maps (inverse accrual maps) across time steps having similar direction. This significantly reduces the number of shadow accrual map (inverse accrual map) computations. Figure 4.12c plots the time taken to accumulate shadows over multiple days, accumulating for 6 hours each day. The advantage of the graph becomes apparent with increasing time periods. For example, when accumulating over a year, the brute force approaches would accumulate shadows for $365 \times 6 = 2190$ hourly intervals. Using the direction graph with a 5° clustering bound, we only need to compute the maps corresponding to 299 edges.

Table 4.2: Memory required by the different approaches for varying resolutions. For shadow map baseline and shadow accrual map, the shadow map resolution is the same as the rendering resolution.

Resolution	**Shadow Map Baseline**	**Shadow Accrual Map**	**Ray Tracing Baseline**	**Inverse Accrual Map**
512×512	1 MB	60 MB	1 MB	14 MB
800×800	2.45 MB	146.5 MB	2.45 MB	34.2 MB
1024×1024	4 MB	240 MB	4 MB	56 MB
1600×1600	9.77 MB	586 MB	9.77 MB	136.7 MB
2048×2048	16 MB	960 MB	16 MB	224 MB

Memory Requirements. Since we are using $n = 60$, shadow accrual maps require storage equivalent of 60 shadow maps. Thus, when using a shadow map resolution of 1024×1024, 240 MB of GPU memory is used by shadow accrual maps (depth values stored as 4 byte floating points). In case of inverse accrual maps, the additional memory required is directly proportional to the the number of source levels that are used. Recall that for a given point, storage is required for mapping the point along both t_1 to t_n and t_n to t_1 for each source level. Since the points are on a plane, the mapped points can be represented using only 2 coordinates. Thus, with the number of source levels $l = 3$, inverse accrual maps require 48 bytes of storage per pixel. Additionally, to maintain the accumulation, it also requires $n = 60$ bits per pixel. Thus, when rendering shadows at a resolution of 1024×1024, inverse accrual maps require approximately 56 MB additional storage. Table 4.2 lists the memory required by the different approaches when using different resolutions.

4.8 Case Studies

In this section, we demonstrate the application of the Shadow Profiler system through two case studies in New York City, a dense urban environment where the impact of new development on streets and parks is a constant concern. The first case study analyzes the impact of new development, more specifically skyscrapers, bordering on Central Park. The second compares various neighborhoods around NYC, specifically looking at desirable shade relative to determinable shadow. Both case studies engage a variety of stakeholders from the general public and advocacy groups to government agencies, such as the City Council, the Department of Parks, and the Department of City Planning.

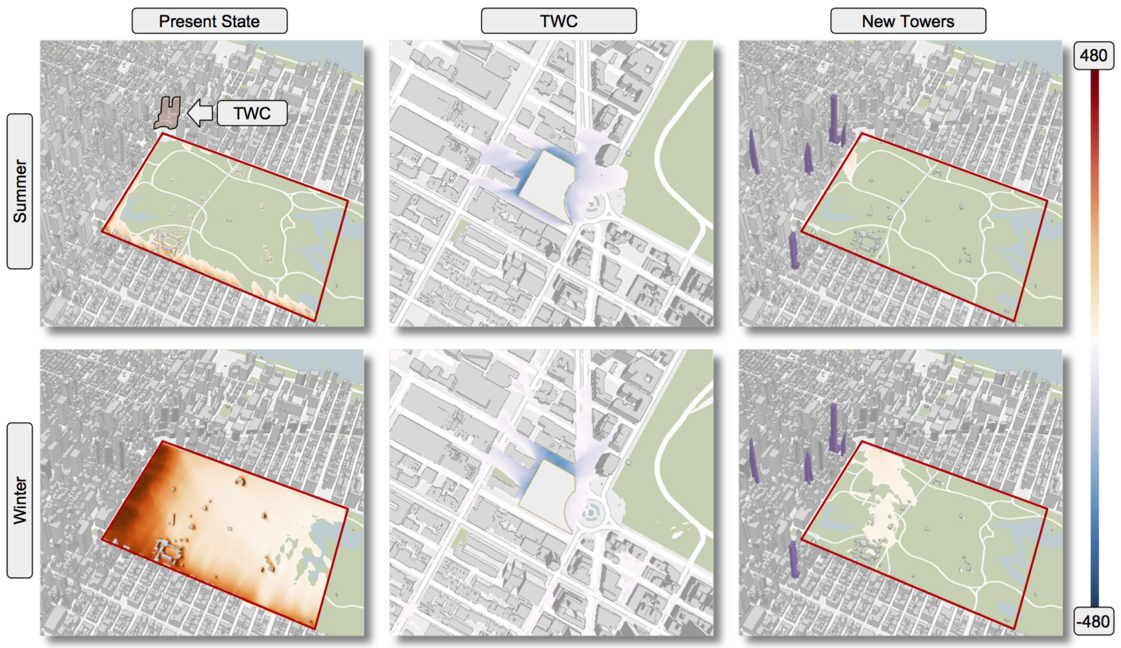


Figure 4.13: Testing the impact of skyscrapers that are under construction south of Central park. Shadows cast during summer and winter with the current state (left). The impact of Time Warner Center (TWC) is used as a baseline for comparison (middle). The impact of the new towers (right). Here, we are visualizing only regions having an impact (positive as well as negative) greater than 30 minutes.

4.8.1 Impact of Buildings on Central Park

Just south of Central Park in Manhattan, a new generation of slender, supertall skyscrapers have begun to rise. There are seven skyscrapers recently built or under construction that range between 780 ft. and 1,490 ft. Their city-wide visibility and proximity to Central Park have raised concerns over shadows cast on the park. With this have come calls to revise NYC zoning regulations to include special review for new towers over 600 ft. [167]. However, there has been little analysis of cast shadow done to test the impact. Whatever analysis that was done was only over fixed time instants [168], which can be misleading since slender towers cast long shadows that move quickly. Comfort level for park-goers and impact on plant life is dependent on the duration of shadow. The longer a person is in shadow the cooler it gets; and plants need a certain number of hours of direct sunlight to grow. So shadows can be both beneficial as well as detrimental depending on the context.

Impact of the Proposed Towers. In this study, we analyze the impact of the skyscrapers south of Central Park by differentiating between negative and beneficial shadows (shade) and using it to compare their performance with shorter and wider buildings. This can also inform a broader discussion on the development and regulation of supertall development in NYC. We divide our analysis time range into two periods representative of negative and beneficial shadows: November and December vs June and July. For consistency we consider an 8 hour period from 8:00 AM to 4:00 PM for each day. For these time periods we first analyze the shadows present in the current context without the seven new skyscrapers, shown in Figure 4.13(left). The region of interest is highlighted in the figure and the gross shadow is visualized. We see that the shadows behave as expected – a lower angle of the sun in winter causes shadows to cover the entire analysis area, while a higher angle in summer results in a tighter shadow area. Note that even though buildings, whether tall or short, cast long shadows at low sun angles (mornings and evenings), its contribution to the overall accumulation is small as reflected in the visualization.

As a baseline for comparison, we analyze the impact of Time Warner Center (TWC) on Central Park. TWC is a skyscraper which is famous for having its design reworked after protests about the shadows it could cast [169]. This is accomplished by removing the tower and computing the impact. Even though the shadows due to TWC cover a large area, the actual area of shadow *only because of TWC* is much smaller. Figure 4.13(center) visualizes the region that is impacted by more than 30 minutes by TWC. Note that this is indeed a small region in its immediate neighborhood extending a little on the north of the building.

Next, we examine the impact of the seven new towers. Since the new cluster of tall towers are located in a wide area south of Central Park they affect a very large area of the park. However, when we restrict to areas impacted by an increase of over 30 minutes of shadows (Figure 4.13(right)), we notice that this is a small fraction of the effected area. Note that this area while comparable for summer, is greater in winter than the concentrated impact that TWC had at Columbus Circle.

Testing Alternate Scenarios. Finally, we use Shadow Profiler to test alternate development scenarios for the new skyscrapers. We modeled a new set of towers, all with the same area as the current seven, with larger floors and lower overall heights. For building lots that allowed it, we doubled the floor plate size. For

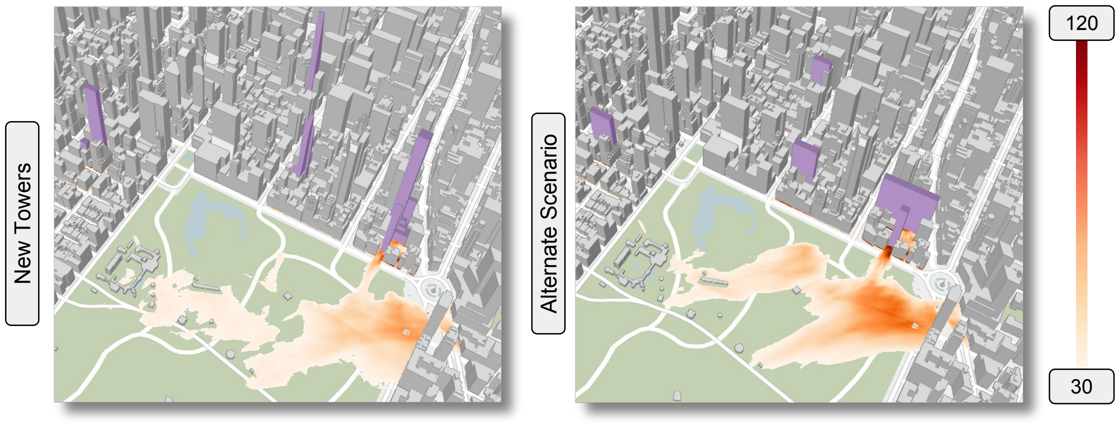


Figure 4.14: We compare the impact of shadows from new and under construction set of skyscrapers in Manhattan (left) with an alternate scenario having shorter towers but with the same total area (right). The towers are highlighted in purple and the impact is visualized using a color map. Note that the impact is stronger due to the shorter towers.

others, we used the largest floor plate size that could be accommodated on the corresponding building lot. These allow us to test if height is indeed an issue that needs to be regulated. This resulted in shadows that are comparable in summer, but having a shorter spread in winter as shown in Figure 4.14. However, notice that quantity of impact (increase in gross shadow) is greater for the short but broader set, especially closer to their base. In fact, when using Boston's shadow duration regulations and considering the area impacted by greater that 60 minutes of new shadow, the shorter towers have greater impact compared to the proposed set.

Thus, choosing the right height with respect to shadow impact is essentially a trade-off between distribution and concentration. That is, given buildings of similar density, a taller building distributes its shadow further away with lower impact over that region, whereas a shorter one concentrates its impact over a smaller area. Given the contention of tall towers' effect on the southern portion of Central Park, this compromise of building height and shadow concentration is particularly important.

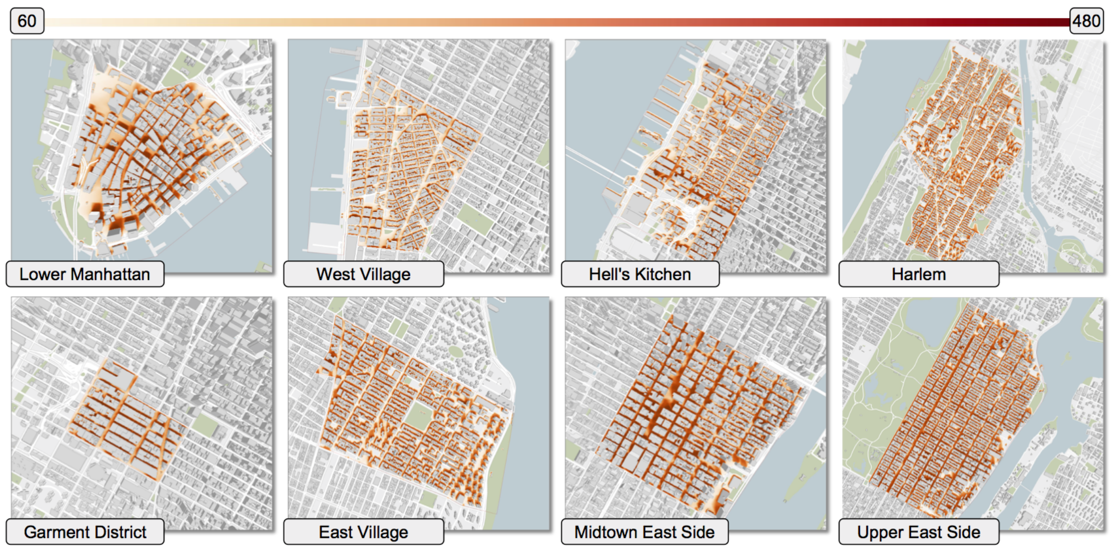


Figure 4.15: Visually comparing year long gross shadows for different neighborhoods in Manhattan. The color map is set to visualize regions with gross shadows greater than an hour.

4.8.2 Citywide Shade Vs. Shadow

City governments are tasked with preserving and promoting the quality of streets and public spaces. While daylight is protected as part of this, through zoning bulk regulations that dictate maximum buildings heights and setbacks, shadows are not rigorously controlled. As a result, a vast majority of new developments are never evaluated on shadows. For projects that do warrant an evaluation, as mentioned earlier, it is only on a small scale primarily because existing tools are prohibitively expensive to scale up the analysis. Shadow Profiler, on the other hand, can allow city planners to analyze shadows comprehensively across the city and appropriately frame policy. In this case study, we first analyze the following neighborhoods for shadows over the entire year: Financial District, West Village, East Village, Garment District, Midtown East, Hell's Kitchen, Upper East Side, and Harlem. As before, the shadows are accumulated for 8 hours per day. This comparison between neighborhoods, illustrated in Figure 4.15, reveals that most neighborhoods in Manhattan are in shadow for more than half the day on average over the entire year. This is expected given that Manhattan has a dense, heavily built urban grain. Closer examination of each neighborhood reveals that the concentration of

Table 4.3: Weights assigned to different months to characterize shade (desirable) vs. shadow (undesirable).

Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
-1	-1	-0.5	0	0.5	1	1	1	0.5	0	-0.5	-1

shadows on streets and sidewalks correlates to the neighborhood's zoned density. However, we find that wide streets, plazas, and parks have relatively lower shadows implying that such places are generally protected against excess shadows through controlling building densities and heights at these locations. For example, there is on an average less than an hour of gross shadow in the park in the center of East Village.

We next select three neighborhoods of interest in Manhattan – West Village, Upper East Side, and Midtown, and analyze them to comprehensively understand how the built contexts relate to their experience through shade versus shadow. While city regulation identifies times when shadow is undesirable [170], given NYC's climate, shade produced during the hot summer months is also highly desirable. To make this distinction, we assign a positive weight for shade, i.e., shadows during summer, and a negative weight for shadows during winter. Table 4.3 shows the weights assigned to different months of the year. This weighing scheme is used to compute the shadow score of these neighborhoods over a period of one year.

Figure 4.16 visualizes the overall score computed over the entire year for the three neighborhoods. Regions with a positive score are shades of blue, while those with a negative score are shades of red. The figure also shows the monthly distribution plots of shadow area (orange plot) and shadow score (blue plot). A region has an overall positive score if it is in shadow for a longer duration in summer than in winter (i.e., more shade than shadow). However, such locations only exist sporadically in lower building density regions of the neighborhoods. This basically indicates that buildings typically have a negative impact with respect to pedestrian comfort levels.

Looking back at Figure 4.13, we see that such a behavior is true even for Central Park, where there is a higher concentration of shadows in winter than in summer. However, these situations for parks are generally mitigated by planting trees and other landscape features. More importantly, through analysis such as

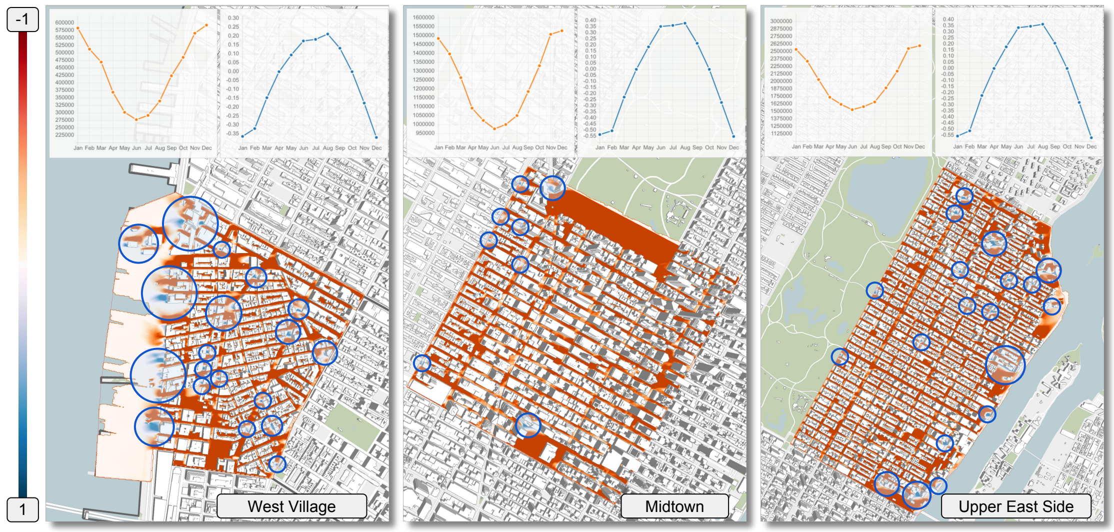


Figure 4.16: The overall effect of shadows on 3 popular Manhattan neighborhoods is mostly negative. Regions with positive yearly score, highlighted by the blue circles, are sparsely distributed in low building density areas. Note that the areas of shadows typically decrease during summer months (orange plot) thus contributing less to the overall score.

the one above it becomes possible to identify problematic regions and suggest corrective measures. It can also be used by city planners to strategically incentivize new development for positive contributions to environmental quality as well as for designers to respond early to these objectives prior to civic review.

4.9 Discussion

Inverse Accrual Maps on Arbitrary Topography. Inverse accrual maps limit the use of computing accurate shadow measures to flat surfaces. This could be overcome through an hybrid approach—use inverse accrual maps for the ground, and resort to brute force ray tracing for buildings. While this can help in a city like NYC which is mostly flat, it will not be as helpful over arbitrary terrains. We plan to explore other approaches, including the use of Monte Carlo ray tracing, to efficiently accumulate shadows in such situations.

Global Illumination. Our current focus is on shadows due to direct sunlight. Public spaces such as parks are typically large enough that the global illumination effects due to the façade materials of the buildings is minimal. However, this will not hold when considering streets / plazas surrounded by towers which have a glass façade. In the future we plan to add functionality to support such scenarios.

Extending Other Shadowing Techniques. Any of the existing shadow maps or shadow volume based techniques can be extended to accumulate shadows by using the linear movement property to compute the shadows at intermediate time steps. This would primarily require computing the shadows for the first and last time steps within the given range (e.g., 1 hour), and appropriately using the interpolated values for the time steps in between.

Conclusions. In this chapter we proposed two techniques, shadow accrual maps and inverse accrual maps, to efficiently accumulate shadows over time. The key in our approach was to implicitly track shadows based on the movement of the sun. These techniques were then used to develop an interactive visual analysis tool called Shadow Profiler. Through using Shadow Profiler to understand how different building types cast shadows, city planners can design zoning regulations to meet their goals while maximizing density and preserving public space quality. It can also function as a learning tool for the general public to understand the effect of shadows on cities [171]. We believe our framework is a first step to change the current planning practice by facilitating the transition from prescriptive rule-based zoning to performance-based zoning, and from discontinuous, isolated, and periodic nature of environmental review to functional continuous relationships between climate and city bulk regulations.

Chapter 5

Image Data: Exploring the Myriad Visuals of a City

The understanding of the city is usually done through the quantification of a subset of attributes from interesting urban data, such as taxi trips [23], stationary sensors (Chapter 2), social media (Chapter 3) or geometry data (Chapter 4). These analyses, however, are mostly limited to non-visual tabular or geometric data, and, while capturing certain aspects of the city, fail to capture the *visual appearance* of urban centers at certain times or conditions. Consider, for instance, snow precipitation during winter, a measure that is clearly quantified in weather data sets: it is one thing to measure 6 inches of snowfall, it is another to *visualize* city streets covered with snow.

An immediate source of visuals of a city is images. The power of visual images is well known; it is why newspapers and websites are full of images. They can encapsulate the spatial and temporal context, as well as make otherwise abstract ideas relatable to different audiences. By using a dense collection of images, it is possible to visualize not only the different blocks, neighborhoods and boroughs of a city, but also its visual changes over days of the week or seasons. Several private companies have been collecting street-level images with the use of cameras mounted on top of cars. Perhaps the most popular, Google Street View [172] allows the exploration of street-level images, emphasizing the particularities of a place rather than cartographic abstractions [173]. Prior works have used these street-level images to predict street safety [174] and detect urban attributes [175], but have not considered the temporal attribute intrinsic to images.

Our goal in this chapter is to allow the interactive exploration of the different visuals offered by a city. We make use of a spatially and temporally dense collection of photographs gathered by a private company in New York City, in combination with publicly available urban data sets. To guide the exploration of this large data set, we use different urban data, enabling the exploration of city views under different constraints, such as weather.

Visuals of a City. In this chapter, we focus on a central element of urban planning to explore the city visuals: a *city block*. Planned cities, such as New York City and Chicago are designed following a grid plan, with rectangular blocks with size usually ranging from 100m to 200m. The importance of blocks (and their size) is so great that Jacos, in *The Death and Life of Great American Cities*, has pointed out that it is one of the generators of diversities in cities [176]. Therefore, the myriad of visuals of a city can be accurately represented by the views of street blocks, in their full diversity and differences.

The identification and extraction of visuals of a city poses several challenges, however. First, we need to efficiently manage and query a collection of images large enough to be representative of the spatial and temporal differences inherent to a city. In our work, we use a collection of 40 million photographs captured by cars around New York City for the period of a year. Unlike Google Street View, however, such collection was gathered using off-the-shelf mobile phones mounted on top of vehicles (see Figure 5.1), without any specialized hardware and no guarantee that consecutive photographs were taken considering a fixed distance.

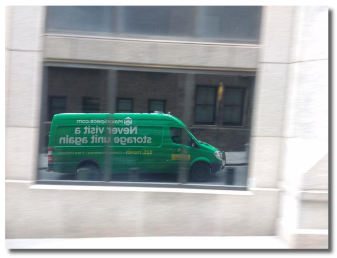


Figure 5.1: Reflection of a laundry pickup car used to capture images in New York City.

Second, we need to allow for the exploration of such large amount of panoramas. This is possible by using popular urban data sets to guide in the exploration flow. For example, weather data can be used to select images during snow days, and this selection can be used to compare against hot summer days. The ability to marry a visual depiction of a city with quantified metrics from urban data sets can be helpful to validate data or to engage stakeholders and residents of a particular region in a city.

Contributions. In this chapter, we take the first steps in designing a system for the exploration of city views considering a large collection of unstructured photographs as well as urban data sets. Our system is composed of two components: a database able to support the querying of a large collection of images and urban data; and a visual interface named *Image Explorer* that allows the exploration of images, together with popular urban data sets. We present a set of case studies that highlight the usefulness of our proposal.

5.1 Related Work

Two important aspects of our investigation are the querying of street-level images and also its visualization and analysis. Next, we review past work related to these topics, highlighting their difference with our current proposal.

Image Organization. The querying of large collection of images can be broadly divided into the ones based on the visual content of the image, or the ones based on some metadata of the image (e.g., keywords, time, location) [177]. In the first case, images are indexed by its visual features, either low-level ones, such as color [178] or texture [179], to high-level features, such as the ones from neural networks [180, 181, 182]. Zhou et al. [183] presented a detailed survey on content-based image retrieval.

Our work follows the path of metadata-based image retrieval, where each image is indexed based on its spatial position, time, and urban data of interest. Several recent papers propose different indexing techniques in order to speed up queries over spatiotemporal data [184]. Here, we make use of a recent technique specifically designed for urban data [185, 186], enabling interactivity in the exploration of the data set.

Also related to our proposal, Barnes et al. [187] proposed a technique to find correspondences between regions of images. Given two images, it computes a distance function between square regions via random sampling. This idea has been applied in several domains, such as image inpainting [188, 189], mimicking artistis styles [190]. Recently, Barnes and Zhang [191] presented a complete survey on patch-based methods. Considering the unstructured nature of the street-level images at hand, we use PatchMatch to group images by their similarity.

Street-level Image Visualization and Analysis. One of the most famous examples of street-level image visualization is Google Street View [172] which allows users to explore cities through *bubble images*, captured by specifically designed cameras. That system was further extended in Microsoft's Streetside [192] to view the side of city blocks from a large distance. While these methods allow for an immersive experience, they require careful capture of spherical images and accurate geotagging. Furthermore, they are not temporally dense, failing to capture the changes over time of city blocks. We propose to use a spatially and temporally dense set of images, captured by inexpensive cameras.

The availability of street-level images has created an opportunity to analyze the city from a new perspective. These images have been used to assess urban environment [193, 194], predict street safety [174], urban change [195], summarization of city landscapes [196], skyexposure models [197], as well as detection of urban features [175, 198, 199, 200, 201]. Arietta et al. [175] presented a method that uses street-level images to identify relationships between the visual appearance of a city and its attributes. More recently, Shen et al. proposed StreetVizor [202], using Google Street View images to analyze urban forms. Recently, Sakurada et al. [203] used a collection of images and mapping data to detect changes in buildings, applying their method to cities damaged by tsunami in Japan. In our work, we propose to take the first steps into exploring and visualizing a temporally dense collection of street-level images. Our system is orthogonal to the previous proposals, and can be used as a backend in studies that want to explore the temporal component of the data.

5.2 Street-level Images

In this section we present an overview of the street-level image data set provided by Carmera, a private company based in NYC[1]. Carmera is equipping cars with a set of four inexpensive mobile phones, each one facing a particular direction (i.e., front, back, left, right). As the cars travel the boroughs of NYC, the cameras from the mobile phones capture images at a regular time interval. Every image is accompanied by metadata, such as the approximate latitude and

[1]The street-level images in this dissertation have been edited to address privacy concerns.

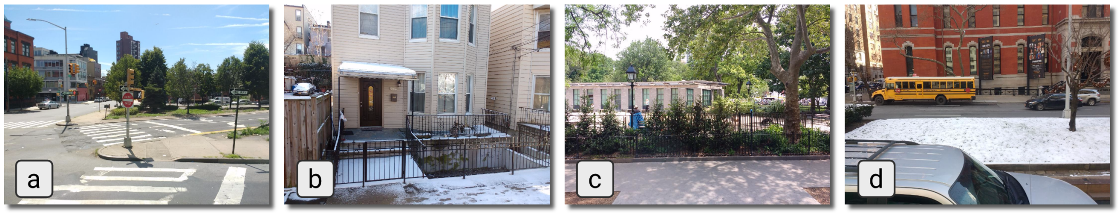


Figure 5.2: Examples of images from the Carmera data set. **(a)**,**(b)** Hot summer day and cold winter day in Brooklyn. **(c)**,**(d)** Spring day and winter day in Manhattan.

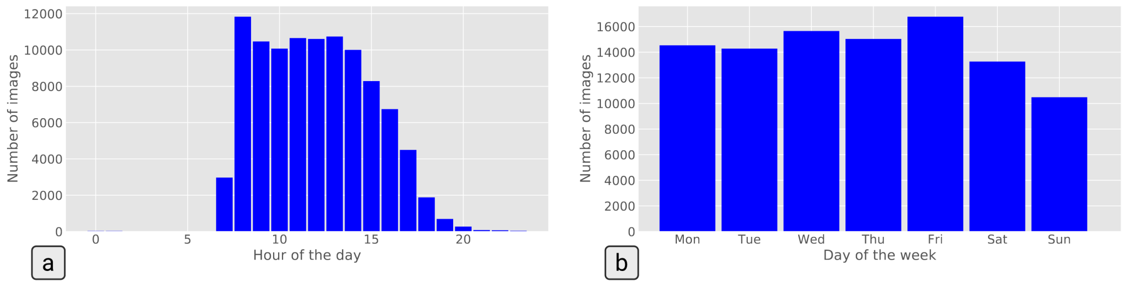


Figure 5.4: Temporal distribution of street-level images throughout hours of the day **(a)** and days of the week **(b)**.

longitude of the car when the image was taken, time and camera orientation. It is important to note that, unlike Google Street View, which deploys cars with the specific goal of capturing street-level images, Carmera is equipping cars whose main purpose is not capturing images (e.g., laundry pickup cars). Because of this, there is no control over illumination, weather, traffic condition or vehicular speed (see Figure 5.2 for image examples).

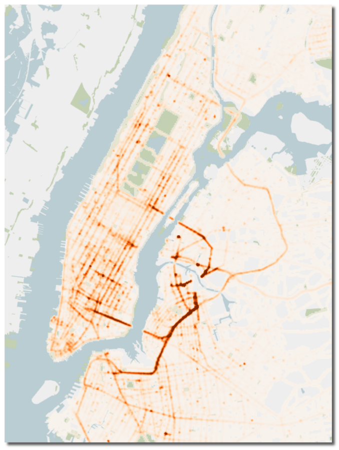


Figure 5.3: Spatial distribution of Carmera's street-level images in New York City.

Given the inexpensive method of image capturing, Carmera's street-level images data is more spatially and temporally dense than other data sets, such as Google Street View or Bing Streetside. Figure 5.3 shows the spatial distribution of images, and Figure 5.4 presents the distribution of images throughout the hours of the day and days of the week. For this work, we use images from March 2016 to February 2017, totaling 52,246,877 images (40 TB in total).

5.3 Visuals of a City

Our goal in this work is to allow the interactive exploration of large collections of street-level images. In order to constraint the number of images and guide users in this exploration, we propose to make use of the growing number of urban data sets, many of them having a spatial or temporal component. Our main idea here is to associate each image with data points that fall within a certain radius from the spatial position of the image.

Consider, for instance, that we would like to visualize the views of Times Square throughout summer. Using a SQL-like syntax, this can be posed as the following query:

$$\textbf{\textit{select}}\ \textit{imgs}\ \textbf{\textit{where}}\ \textit{season}\ \textbf{\textit{is}}\ \textit{Summer}\ \textbf{\textit{and}}\ \textit{location}\ \textbf{\textit{in}}\ \textit{Times Square}$$

Another possible query is to compare Times Square during a hot summer day and a snowy winter day. This can be posed as the following queries:

$$\textbf{\textit{select}}\ \textit{imgs}\ \textbf{\textit{where}}\ \textit{season}\ \textbf{\textit{is}}\ \textit{Summer}\ \textbf{\textit{and}}\ \textit{temp} > 30\ \textbf{\textit{and}}\ \textit{pos}\ \textbf{\textit{in}}\ \textit{Times Square}$$
$$\textbf{\textit{select}}\ \textit{imgs}\ \textbf{\textit{where}}\ \textit{season}\ \textbf{\textit{is}}\ \textit{Winter}\ \textbf{\textit{and}}\ \textit{snw} > 0\ \textbf{\textit{and}}\ \textit{pos}\ \textbf{\textit{in}}\ \textit{Times Square}$$

Note then that our system must be able to interactively solve queries of the following type over an input of images (meta data) and urban data:

$$\textbf{\textit{select}}\ \textit{imgs}\ \textbf{\textit{between}}\ t_1\ \textit{and}\ t_2\ \textbf{\textit{where}}\ \textit{constraints}\ C$$

where $t1$ and $t2$ specify the time period of the data to consider. The constraints $C = \bigcup_r \{C_r\}$ define the constraints over the spatio-temporal domain and also over all urban data considered (e.g., weather information, crime, tweets). C_u specifies a set of values for urban data set u that have to be satisfied. For example, if we want to query images from when the temperature is greater than 30°C, then $C = \{C_{temperature} = \{temp > 30°\text{C}\}\}$. Alternatively, if we want only images that have more than two crime reports within 100 meters and within an one hour window of the time that the picture was taken, we would have $C = \{C_{crime} = \{within(100, 1) > 2\}\}$, where $within(\epsilon, \alpha)$ returns the number of crime occurrences within ϵ meters of the image location, and within α hours of its time.

In this section, we describe the main components of our system, and discuss its properties. We also describe extensions that enable additional features such as the possibility to merge images, and compute time lapses.

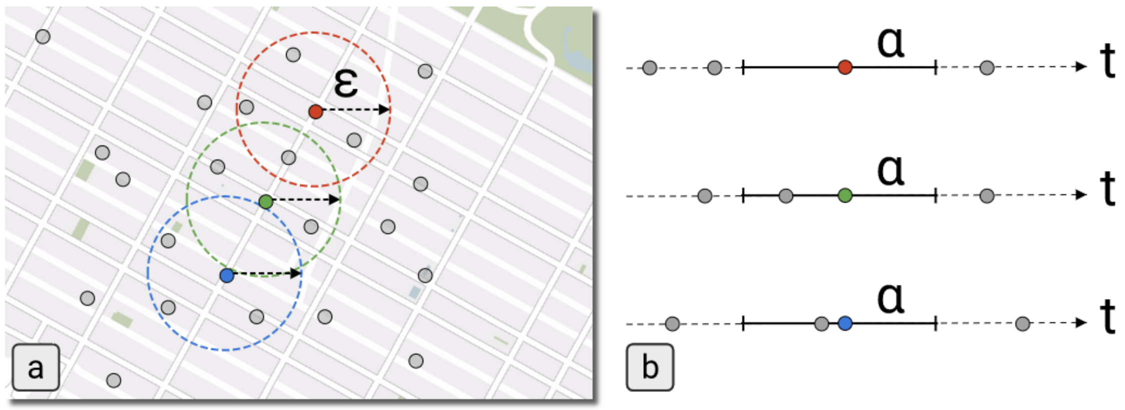


Figure 5.5: Joining images (red, green and blue dots) and urban data (gray dots). For each image, we associate the data within ϵ meters **(a)** and within α hours **(b)**.

5.3.1 Joining Images and Urban Data

In order to support queries as the ones previously described, we perform a spatio-temporal join between the street-level images I and each urban data set D as a pre-processing step. Our system supports data sets that are composed of time series (e.g., weather, SPL data described in Chapter 2), lists of latitude and longitude coordinates (e.g., social media data as in Chapter 3) or as a 2D grid (e.g., shadow accumulation as in Chapter 4). In the case of a join between I and a temporal data set D, for each image $i \in I$ at timestamp t_i, we simply associate value d at timestamp t_d, where $t_i = t_d$. In the case of a join between I and a spatio-temporal data set D, for each image $i \in I$ we find the points $d \in D$ that are within ϵ meters of the position of the image (Figure 5.5(a)). For each d that satisfies this constraint, we verify if the timestamp of d is within α hours of the timestamp of i (when the image was captured) (Figure 5.5(b)). Alternatively, in a join between I and a 2D grid D, for each image $i \in I$, we find the grid cells within ϵ meters of i and apply an aggregate function on its values (e.g., average, sum, max). We augment our image database with the result of the join computation for all data sets considered.

Querying. Our system stores the images metadata and the result of the join computations in a spatio-temporal database, while the images itself are stored in secondary memory. After the query is computed, the images are retrieved from disk.

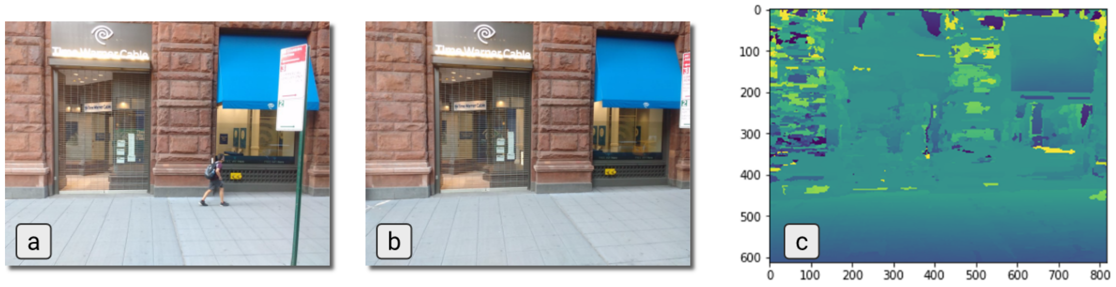


Figure 5.6: PatchMatch when considering two consecutive images (**(a)** and **(b)**). **(c)** shows the result, with the distance between patches.

5.3.2 Image Matching

The result of the queries is a collection of images, grouped by car, along with their approximate location and direction. We call each group a “strip”. Given the low accuracy of the latitude and longitude information, as well as the lack of sampling frequency, we propose to perform an additional optional step in order to align the strips. We process each strip by performing PatchMatch [187] on each neighboring pair of images within each strip. This gives us a distance function between patches of consecutive images. If the overall distance is below a certain threshold, we consider that the images are too similar and that one can be discarded. This is especially important when considering a data set like the one used here. More often than not a car used to capture images will not move for a long period of time, resulting in images that are indistinguishable; by running PatchMatch we can effectively remove these similar images and therefore the visual clutter.

PatchMatch. The PatchMatch [187] algorithm attempts to find corresponding patches between a pair of image. For each pixel on the first image, PatchMatch compares the patch centered on that pixel to patches in the other image, attempting to find the best match. It accelerates the search by both examining random patches, as well as ones based on neighboring pixels in the first image.

PatchMatch result is a dense correspondence between the images (see Figure 5.6). For our purposes we found that PatchMatch produced better correspondences than the invariant features often used on structure from motion. We note that, by using PatchMatch, we loose the ability to compare features between more than two images.

For our application, we slightly modified the PatchMatch algorithm to run on a GPU. Instead of starting from the top left and going row by row, column by column, we examine every pixel in the first image in parallel. To help accelerate the diffusion of information, we run 4 stages without random samples in-between each full iteration. In our system, we use the result of PatchMatch to discard images that are too similar (i.e., average distance between patches is small), reducing the visual clutter when a query returns images that are indistinguishable from each other.

5.4 Image Explorer

In this section, we describe our interface, Image Explorer, designed for the interactive exploration of street-level images.

5.4.1 Desiderata

The main tasks we are interested while exploring street-level images are: 1) compare the same region, but under different constraints; and 2) compare different regions, under similar or different constraints. To accomplish this, we develop a web-based prototype that satisfies the following requirements: 1) specify, execute, and visualize spatio-temporal queries; and 2) ability to select and compare multiple regions across time; and 3) visualize the images from a given region and time. We now briefly discuss the interface details, followed by a description of the backend query system that is handled by a server.

5.4.2 Visual Interface

The Image Explorer interface consists of two main components: a query configuration panel and an image groups viewer (see Figure 5.7). Next, we detail each component of our visual interface.

Configuration Panel. This panel (Figure 5.7(right)) allows the user to specify the queries as a set of spatiotemporal constraints. The user first specifies a spatial region of interest, followed by a time range. In order to add constraints based on urban data, the user adds the constraint in the format $constraintname, min, max$, where $constraintname$ is the column of interest (e.g., temperature) and min, max is the range (e.g., 30°C,40°C).

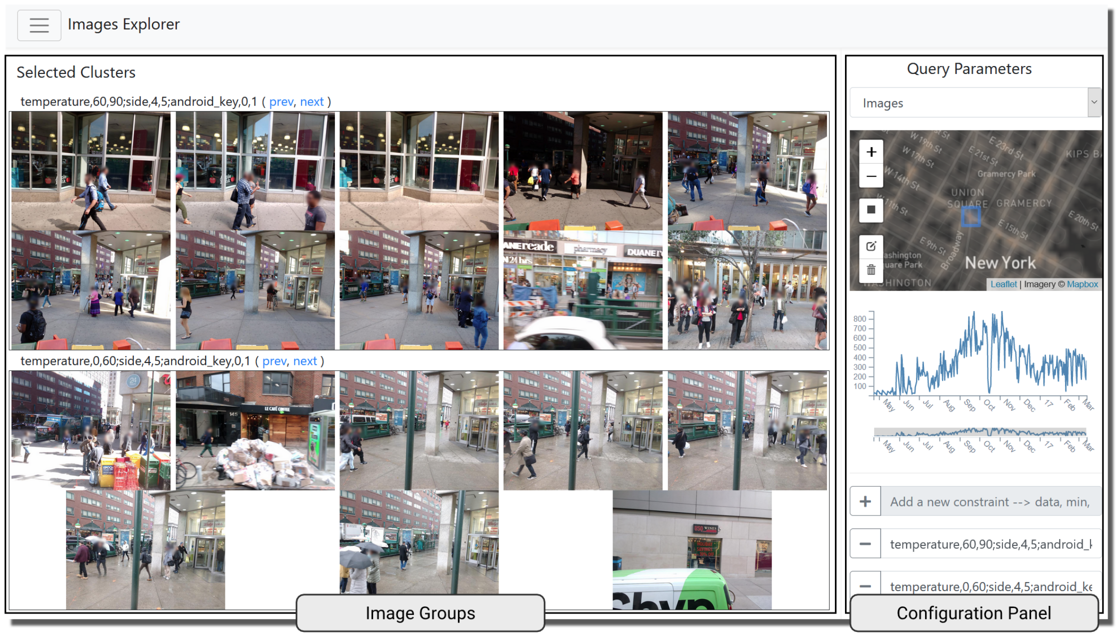

Images Explorer
Selected Clusters
temperature,60,90;side,4,5;android_key,0,1 (prev, next)
temperature,0,60;side,4,5;android_key,0,1 (prev, next)
Image Groups
Query Parameters
Images
New York
Add a new constraint --> data, min,
temperature,60,90;side,4,5;android_k
temperature,0,60;side,4,5;android_ke
Configuration Panel

Figure 5.7: Image Explorer overview. The user can specify a query using the Configuration Panel. The resulting images are displayed in the Image Groups Viewer.

Image Groups Viewer. Using the Configuration Panel, the user can group one or more groups of images that will be displayed in the Image Groups Viewer (Figure 5.7(left)). Each group is composed of images that satisfy the constraints and ordered by the time when the image was captured. In other words, the left-most image will have a timestamp t_0, the right-most image will have a timestamp t_1, and $t_0 < t_1$. In order to minimize the visual cluttering, we restrict the number of images displayed to be proportional to the available screen space, thus avoiding unnecessary retrieval of images. A widget on top of each group allows the user to navigate through the series of images returned by the query, effectively navigating through the street in the direction the car is moving.

5.4.3 Query Backend

We implemented a C++ server-based backend in order to allow easy access to the visual interface through a web browser. The queries are submitted to the server as HTTP requests and, once computed, the results are sent to the requesting browser using a JSON format. The images are encoded as base64 strings and compressed using the JPEG format. Even though the encoding increases the size of file image by about one third (when compared to the original non-base64 JPEG file), it ensures easy compatibility between different browsers. In order to allow for an interactive experience while exploring street-level images, we make use of a database specifically designed for spatial-temporal queries [185, 186].

5.5 Case Studies

In this section, we exemplify how our system can be used in the visual analysis of a large collection of street-level images. The first case study goes back to a case study previously presented in Section 4.8.2 in order to better understand the impact of shadows in a dense neighbourhood of NYC. The second case shows how images can be used to further enhance the study of urban noise. Finally, the third case study explores how the system can be useful when exploring the development of a city region. The case studies highlight the importance of the visual element when understanding not only urban data, but also the city itself.

5.5.1 Shadow Impact

In Section 4.8.2, we highlighted the difference between positive shade during summer and negative shadow during winter. As can be seen in Figure 4.16, there are only a few regions in NYC where the amount of shade is actually greater than the amount of shadow. In this first case study, we use our system to further analyze one of these regions, the northwest corner of Bryant Park. Our goal here was to better understand how this particular region looks when in shade or shadow.

We selected the region of interest and queried for all images captured during summer and winter in three periods of the day. We filtered out images from cloudy or rainy days, therefore guaranteeing that shadows were being cast by buildings. Figure 5.8 shows some of the images that satisfied these constraints. We can notice here that, as pointed in Section 4.8.2, this region does indeed have a greater amount of shade than shadow. The power to marry a quantitative analysis with views from the city can be important to engage different groups and illustrate the problem in a much more relatable way, making it easier to drive the interest of stakeholders or the community itself.

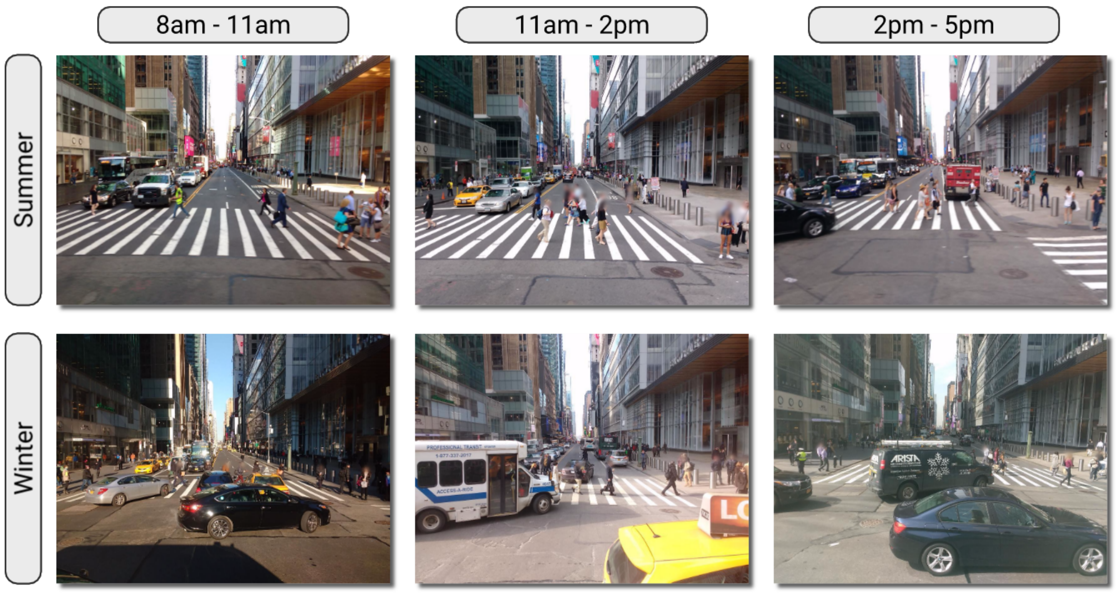


Figure 5.8: The street-level images were captured in three different periods of the day during winter and summer, and show one of the few regions in NYC where the amount of shade is greater than shadow.

5.5.2 Noise Complaints

The problem of urban noise was studied in Section 2.5.1, highlighting how the passive sensing of decibel level can greatly benefit the enforcement of the noise code in cities. Here, we use the street-level images to further assist in the analysis of noise, especially noise from construction.

In NYC, residents can file a noise complaint to the 311 service [204], detailing the origin of the noise and also the time it happened. This information is used by city agencies, such as the Department of Environmental Protection, to schedule noise enforcement visits and issue fines when appropriate. We used the 311 data

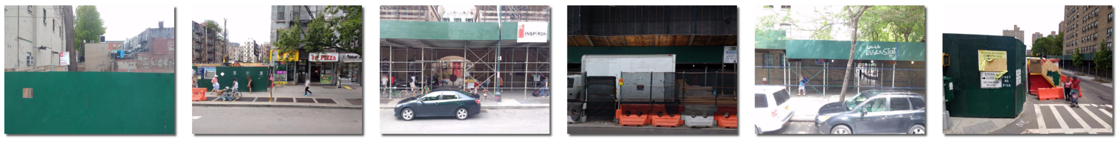


Figure 5.9: Street-level images that were captured within 100 meters and one hour of a noise complaint in NYC. Here we show only images that were near a construction site, which can be used to better guide city agencies when enforcing the noise code.

set to select images that were captured within 100 meters of a noise complaint and also within one hour of the complaint report, considering all boroughs of NYC. Some of the images that satisfied these constraints are shown in Figure 5.9. All images shown are near a construction site, identified by the green boards.

This exploration and the ability to use both a visual representation and also urban data can be used to further assist the agencies responsible for enforcing the noise code. By viewing the development site, the agencies can assess the stage of construction or the presence of heavy machinery, and better schedule the visits of noise enforcement agents.

5.5.3 Development of a City Block

The neighbourhood of Williamsburg in Brooklyn, NYC has been the focus of construction initiatives in recent years, with old houses giving way to modern buildings in a phenomenon that is rapidity changing the look of the region. Our system provides an intuitive way to visually grasp these changes through the course of a year, and analyze the evolution of a construction site from the early stages of development until the inauguration of a new building.

In this case study, we used construction permits issued by the Department of Buildings of New York City to assist in our exploration. We started by filtering the images that were captured within 100 meters of an active construction site. By selecting a street block of Williamsburg, we query for images from April 2016, identifying an active construction site in its early stages of development. We then selected images until March 2017, when a new store was inaugurated. Through the

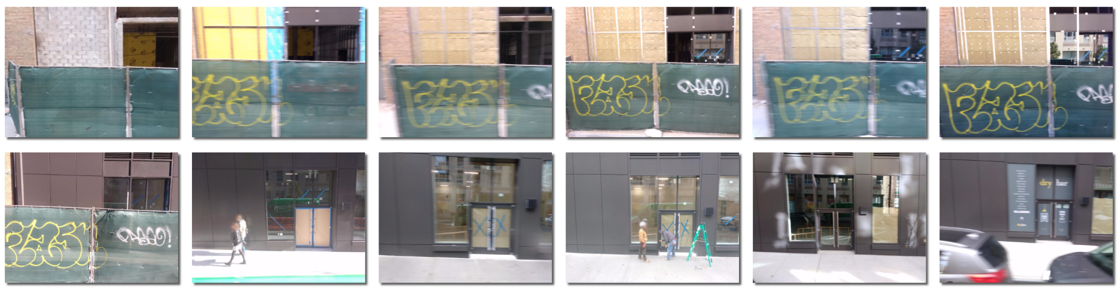


Figure 5.10: Construction of a new development in the neighbourhood of Williamsburg in Brooklyn, from April 2016 (top left) to March 2017 (bottom right), when a new store was inaugurated.

use of street-level images, it is possible to see the entire evolution of the development (Figure 5.10), gradually seeing how this particular city block is changing through the year.

5.6 Discussion

In this chapter, we presented the first steps into exploring a large collection of street-level images. We presented a system that makes use of an efficient backend to drive the interactivity of a web-based tool that allows the exploration of images in conjunction with urban data sets. Using the system, we presented three different case studies, highlighting the usefulness in marrying data and image to further analyze urban problems.

While our current system can drive interactivity in the exploration of this data set, there are a few limitations that we plan to address in future work. The first is the organization of the images itself. In order to simplify the exploration process, we plan to use image processing techniques to assign images to particular buildings; instead of filtering by a certain region, it would be possible to select a building (or lot) of interest and query for all images that show that particular site. This would also address a serious problem with the data set: in regions with a high density of buildings, such as the Financial District, the geolocation accuracy of the images is severely compromised. A process where we assign images to buildings would alleviate this problem.

Considering the recent developments in video compression, we plan to investigate the use of compression methods to minimize the storage requirements, assessing the feasibility of a data structure capable of indexing spatiotemporal images without a large memory overhead. Another interesting direction is the stitching of consecutive images in order to avoid the visual clutter. Stitching techniques have been applied previously in the visualization of street-level images [205], but not considering a temporally dense collection of images. It would be interesting to pursue this line of research in order to minimize the number of images shown to the user and make the exploration process more intuitive.

Chapter 6

Conclusions and Future Work

This dissertation presented four main contributions to the interactive analysis of urban data. The first contribution was Time Lattice [31], a data structure that enables the interactive visual analysis of large time series. We also presented a visual interface that used the structured and allowed domain experts to explore a large set of time series generated from sensors deployed throughout New York City that measure the decibel level at a high resolution. The second contribution was TopKube [32], a data structured that aims to support the interactive computation of spatiotemporal Top-K queries, allowing the exploration of not only where and when things are happening, but also what is happening. The third contribution was Shadow Accrual Maps [36], a data structured that enables the fast accumulation of shadows. Together with the structure, we also proposed a visual interface that was used by domain experts to explore the impact of shadows in a number of scenarios set in New York City. Finally, the fourth contribution of this dissertation was a system that allowed the interactive exploration of a large collection of street-level images, showing how the visual aspect of a city changes over space and time.

The contributions presented here received a great deal of attention from different stakeholders, such as journalists, city residents and city agents. This engagement with the community can lead to the identification of *new* fundamental research challenges, that contribute to a greater understanding of several problems that face cities today. Next, we present some of these challenges and highlight interesting directions of research in the field of visualization.

Adoption by Domain Experts. The contributions of this dissertation were used by domain experts through visual interfaces that were designed with the goal of exploring a specific data set. However, it is common that domain experts have their own pipeline of analysis and preferred tools, and migrating to a new environment might be too cumbersome for them. For instance, many scientists are used with frameworks such as pandas, scikit or languages such as Python and R. Our contributions here, and many of the data structures proposed in the community in recent years, are very specific and do not integrate well (or at all) with popular frameworks and languages. An interesting direction is the investigation of a platform for spatiotemporal urban data, where data scientists could choose between different data structures and indexes for their tasks, but maintain the same high-level abstractions and analysis pipeline offered by their preferred frameworks and languages.

Simulation of Urban Phenomena. In this dissertation, we presented the computation of city-scale shadow accumulation with the use of highly-accurate building models. There are, however, several urban phenomena that are still not analyzed at this scale or accuracy. An example of this is noise propagation; with the availability of data sets describing major sources of noise (e.g., bus, construction, trains), coupled with highly-accurate city models, it would be interesting to model how noise propagates through the urban environment.

Optimizing City Blocks. Procedural modeling is becoming increasingly popular in urban planning applications. The work by Venegas et al. [206] proposed a technique that allows the generation of 3D urban models based on a set of parameters (e.g., sky exposure, distance to park). Doraiswamy et al. [207] recently presented a topology-based approach for the creation of view-enhanced tower designs. An interesting direction to pursue would be to enhance these methods by adding a set of complex and dynamics parameters, such as the shadow accumulation presented here. One can image the design of buildings (or city blocks) that conform to a set of constraints, such as the amount of direct sunlight reaching public spaces.

Appendix A

Additional Shadow Map Experiments

Efficiency of the Shadow Map Baseline. To demonstrate the efficiency of our shadow map implementation used as baseline in our experiments, we compare it with an industry standard game engine. For this comparison, we simulated the same experiment as in Figure 4.11(a) of the dissertation. This experiment computes the average time to render 60 shadow maps (corresponding to each minute of an hour) for different camera positions and sun positions. Using the shadow map baseline, this time varies between 100 ms and 240 ms — the time taken to render shadows for a single time step is between 1.5 ms and 4 ms. In fact, the time to render shadows for a given single time step even when the shadow map resolution is 2048×2048 is still less than 5 ms.

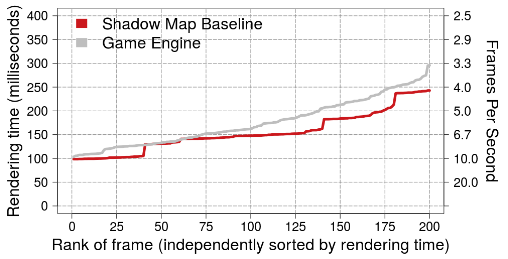


Figure A.1: Comparing our shadow map baseline implementation with shadow computation on an industry standard game engine. The figure plots the average time taken to compute shadows for an hour with a resolution of 1 minute, that is, 60 shadow maps are computed per hour.

Using the game engine, we simply computed shadows for 60 time steps per hour for the same camera positions and sun positions. We however did not perform any accumulation – accumulation will only add an additional step in the game engine pipeline to store these shadows to do the necessary computation. This not only takes up more space, but also requires additional time. Note that even without accumulation being explicitly computed, the performance of our baseline shadow map implementation is on par with that of the game engine (Figure A.1).

Bibliography

[1] Meredith Reba, Femke Reitsma, and Karen C. Seto. Spatializing 6,000 years of global urbanization from 3700 BC to AD 2000. *Scientific Data*, 3, 2016.

[2] Edward L. Glaeser, Hedi D. Kallal, José A. Scheinkman, and Andrei Shleifer. Growth in cities. *Journal of Political Economy*, 100(6):1126–1152, 1992.

[3] UN Department of Economic and Social Affairs. UN world urbanization prospects: The 2014 revision highlights. Available: `http://esa.un.org/unpd/wpp/`. Accessed on: Oct. 23, 2017.

[4] John Snow. On the mode of communication of cholera. *Edinburgh Medical Journal*, 1(7):668–670, 1856.

[5] Max Weber. *The City*. 1921.

[6] Louis Wirth. Urbanism as a way of life. *American Journal of Sociology*, 44(1):1–24, 1938.

[7] Robert E. Park, Ernest W. Burgess, and Roderick D. McKenzie. *The City*. 1925.

[8] Ruth Shonle Cavan. The Chicago School of Sociology, 1918-1933. *Urban Life*, 11(4):407–420, 1983.

[9] Wayne G. Lutters and Mark S. Ackerman. An introduction to the Chicago School of Sociology. *Interval Research Proprietary*, pages 02–06, 1996.

[10] Stanley K. Smith. Toward a methodology for estimating temporary residents. *Journal of the American Statistical Association*, 84(406):430–436, 1989.

[11] Doug Washburn and Usman Sindhu. Helping CIOs understand "smart city" initiatives. *Growth*, 17(2):1–17, 2009.

[12] Vito Albino, Umberto Berardi, and Rosa Maria Dangelico. Smart cities: Definitions, dimensions, performance, and initiatives. *Journal of Urban Technology*, 22(1):3–21, 2015.

[13] NYC Open Data. Available: `https://nycopendata.socrata.com`. Accessed on: Oct. 23, 2017.

[14] San Francisco Data. Available: `https://data.sfgov.org`. Accessed on: Oct. 23, 2017.

[15] Chicago Data Portal. Available: `https://data.cityofchicago.org`. Accessed on: Oct. 23, 2017.

[16] Brett Goldstein and Lauren Dyson. *Beyond Transparency: Open Data and the Future of Civic Innovation*. Code for America Press, 2013.

[17] Luciano Barbosa, Kien Pham, Claudio T. Silva, Marcos R. Vieira, and Juliana Freire. Structured open urban data: Understanding the landscape. *Big Data*, 2(3):144–154, 2014.

[18] Twitter public API. Available: `https://dev.twitter.com/streaming`. Accessed on: Oct. 23, 2017.

[19] Bart Thomee, David A. Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. YFCC100M: The new data in multimedia research. *Communications of the ACM*, 59(2):64–73, 2016.

[20] Juliana Freire, Cláudio T. Silva, Huy T. Vo, Harish Doraiswamy, Nivan Ferreira, and Jorge Poco. Riding from urban data to insight using New York City taxis. *IEEE Data Engineering Bulletin*, 37(4):43–55, 2014.

[21] John W. Tukey. *Exploratory Data Analysis*. Addison-Wesley, 1977.

[22] Remco Chang, Ginette Wessel, Robert Kosara, Eric Sauda, and William Ribarsky. Legible Cities: Focus-dependent multi-resolution visualization of urban relationships. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1169–1175, 2007.

[23] Nivan Ferreira, Jorge Poco, Huy T Vo, Juliana Freire, and Cláudio T Silva. Visual exploration of big spatio-temporal urban data: A study of New York City taxi trips. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2149–2158, 2013.

[24] Nivan Ferreira, Marcos Lage, Harish Doraiswamy, Huy T. Vo, Luc Wilson, Heidi Werner, Muchan Park, and Claudio T. Silva. Urbane: A 3D framework to support data driven decision making in urban development. In *Proceedings of the 2015 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pages 97–104, 2015.

[25] Gennady Andrienko, Natalia Andrienko, Peter Bak, Daniel Keim, and Stefan Wrobel. Visual analytics focusing on spatial events. In *Visual Analytics of Movement*, pages 209–251. Springer Berlin Heidelberg, 2013.

[26] Gennady Andrienko, Natalia Andrienko, Christophe Hurter, Salvatore Rinzivillo, and Stefan Wrobel. Scalable analysis of movement data for extracting and exploring significant places. *IEEE Transactions on Visualization and Computer Graphics*, 19(7):1078–1094, 2013.

[27] Harish Doraiswamy, Nivan Ferreira, Theodoros Damoulas, Juliana Freire, and Claudio T. Silva. Using topological analysis to support event-guided exploration in urban data. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2634–2643, 2014.

[28] Jorge Poco, Harish Doraiswamy, Huy T. Vo, João L. D. Comba, Juliana Freire, and Cláudio T. Silva. Exploring traffic dynamics in urban environments using vector-valued functions. *Computer Graphics Forum*, 34(3):161–170, 2015.

[29] Fabio Miranda, Harish Doraiswamy, Marcos Lage, Kai Zhao, Bruno Gonçalves, Luc Wilson, Mondrian Hsieh, and Cláudio T. Silva. Urban Pulse: Capturing the rhythm of cities. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):791–800, 2017.

[30] Zhicheng Liu and Jeffrey Heer. The effects of interactive latency on exploratory visual analysis. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2122–2131, 2014.

[31] Fabio Miranda, Marcos Lage, Harish Doraiswamy, Charlie Mydlarz, Justin Salamon, Yitzchak Lockerman, Juliana Freire, and Claudio T. Silva. Time Lattice: A data structure for the interactive visual analysis of large time series. *Computer Graphics Forum*, 37(3):23–35.

[32] Fabio Miranda, Lauro Lins, James T. Klosowski, and Claudio T. Silva. Topkube: A rank-aware data cube for real-time exploration of spatiotemporal data. *IEEE Transactions on Visualization and Computer Graphics*, 24(3):1394–1407, 2018.

[33] Hongchao Fan, Alexander Zipf, Qing Fu, and Pascal Neis. Quality assessment for building footprints data on OpenStreetMap. *International Journal of Geographical Information Science*, 28(4):700–719, 2014.

[34] Barron Christopher, Neis Pascal, and Zipf Alexander. A comprehensive framework for intrinsic OpenStreetMap quality analysis. *Transactions in GIS*, 18(6):877–895.

[35] Michael F. Goodchild and Linna Li. Assuring the quality of volunteered geographic information. *Spatial Statistics*, 1:110–120, 2012.

[36] Fabio Miranda, Harish Doraiswamy, Marcos Lage, Luc Wilson, Mondrian Hsieh, and Cláudio T. Silva. Shadow accrual maps: Efficient accumulation of city-scale shadows over time. *IEEE Transactions on Visualization and Computer Graphics*, 2018.

[37] Arline L. Bronzaft and Louis Hagler. *Noise: The Invisible Pollutant that Cannot Be Ignored*, pages 75–96. Springer Netherlands, 2010.

[38] City of New York. New York City Local Law No. 113. Available: `http://www.nyc.gov/html/dep/pdf/law05113.pdf`. Accessed on: Jan. 31, 2018.

[39] City of Portland. City Code, Title 18: Noise Control. Available: `https://www.portlandoregon.gov/citycode/?c=28182`. Accessed on: Jan. 31, 2018.

[40] Charlie Mydlarz, Justin Salamon, and Juan Pablo Bello. The implementation of low-cost urban acoustic monitoring devices. *Applied Acoustics*, 117:207–218, 2017.

[41] Charlie Mydlarz, Charles Shamoon, and Juan Pablo Bello. Noise monitoring and enforcement in New York City using a remote acoustic sensor network. In *Proceedings of INTER-NOISE and NOISE-CON*, pages 5509–5520, 2017.

[42] SONYC: Sounds of New York City. Available: `https://wp.nyu.edu/sonyc/`. Accessed on: Jan. 31, 2018.

[43] Tuomas Pelkonen, Scott Franklin, Justin Teller, Paul Cavallaro, Qi Huang, Justin Meza, and Kaushik Veeraraghavan. Gorilla: A fast, scalable, in-memory time series database. *Proceedings of the VLDB Endowment*, 8(12):1816–1827, 2015.

[44] Andreas Bader, Oliver Kopp, and Michael Falkenthal. Survey and comparison of open source time series databases. *Datenbanksysteme für Business, Technologie und Web (BTW 2017)-Workshopband*, page 266.

[45] InfluxDB. Available: `https://github.com/influxdata/influxdb`. Accessed on: Jan. 31, 2018.

[46] KairosDB. Available: `https://kairosdb.github.io/`. Accessed on: Jan. 31, 2018.

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

[48] Cícero A. L. Pahins, Sean A. Stephens, Carlos Scheidegger, and João L. D. Comba. Hashedcubes: Simple, low memory, real-time visual exploration of big data. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):671–680, 2017.

[49] Maheshkumar Sabhnani, Andrew W. Moore, and Artur W. Dubrawski. T-cube: A data structure for fast extraction of time series from large datasets. Technical report, DTIC Document, 2007.

[50] Luca Deri, Simone Mainardi, and Francesco Fusco. tsdb: A compressed database for time series. *Traffic Monitoring and Analysis*, pages 143–156, 2012.

[51] Maxim Buevich, Anne Wright, Randy Sargent, and Anthony Rowe. Respawn: A distributed multi-resolution time-series datastore. In *Proceedings of the 2013 IEEE 34th Real-Time Systems Symposium*, pages 288–297, 2013.

[52] TimescaleDB. Available: `https://timescale.com/`. Accessed on: Jan. 31, 2018.

[53] Søren Kejser Jensen, Torben Bach Pedersen, and Christian Thomsen. Time series management systems: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 29(11):2581–2600, 2017.

[54] 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 subtotals. *Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.

[55] Chen Chen, Xifeng Yan, Feida Zhu, Jiawei Han, and S Yu Philip. Graph OLAP: Towards online analytical processing on graphs. In *Proceedings of the 2008 IEEE International Conference on Data Mining*, pages 103–112, 2008.

[56] Cindy Xide Lin, Bolin Ding, Jiawei Han, Feida Zhu, and Bo Zhao. Text cube: Computing ir measures for multidimensional text database analysis. In *Proceedings of the 2008 IEEE International Conference on Data Mining*, pages 905–910, 2008.

[57] Kevin Beyer and Raghu Ramakrishnan. Bottom-up computation of sparse and iceberg cube. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pages 359–370, 1999.

[58] Jiawei Han, Yixin Chen, Guozhu Dong, Jian Pei, Benjamin W Wah, Jianyong Wang, and Y Dora Cai. Stream cube: An architecture for multi-dimensional analysis of data streams. *Distributed and Parallel Databases*, 18(2):173–197, 2005.

[59] Qiyang Duan, Peng Wang, MingXi Wu, Wei Wang, and Sheng Huang. Approximate query on historical stream data. In *Database and Expert Systems Applications*, pages 128–135. Springer, 2011.

[60] Jian Zhao, Fanny Chevalier, Emmanuel Pietriga, and Ravin Balakrishnan. Exploratory analysis of time-series with ChronoLenses. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2422–2431, 2011.

[61] Jarke J. Van Wijk and Edward R. Van Selow. Cluster and calendar based visualization of time series data. In *Proceedings of the 1999 IEEE Symposium on Information Visualization (InfoVis)*, pages 4–9, 1999.

[62] Peter McLachlan, Tamara Munzner, Eleftherios Koutsofios, and Stephen North. LiveRAC: Interactive visual exploration of system management time-series data. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1483–1492, 2008.

[63] Waqas Javed, Bryan McDonnel, and Niklas Elmqvist. Graphical perception of multiple time series. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):927–934, 2010.

[64] Sônia Fernandes Silva and Tiziana Catarci. Visualization of linear time-oriented data: a survey. In *Proceedings of the First International Conference on Web Information Systems Engineering*, pages 310–319, 2000.

[65] Wolfgang Müller and Heidrun Schumann. Visualization for modeling and simulation: visualization methods for time-dependent data-an overview. In *Proceedings of the 35th Conference on Winter Simulation: Driving Innovation*, pages 737–745, 2003.

[66] Wolfgang Aigner, Silvia Miksch, Wolfgang Müller, Heidrun Schumann, and Christian Tominski. Visualizing time-oriented data - a systematic view. *Computers and Graphics*, 31(3):401–409, 2007.

[67] Lior Berry and Tamara Munzner. BinX: Dynamic exploration of time series datasets across aggregation levels. In *Proceedings of the 2004 IEEE Symposium on Information Visualization (InfoVis)*, 2004.

[68] Ming C. Hao, Umeshwar Dayal, Daniel A. Keim, and Tobias Schreck. Multi-resolution techniques for visual exploration of large time-series data. In *Proceedings of the 9th Joint Eurographics / IEEE VGTC Conference on Visualization*, pages 27–34, 2007.

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

[70] Prithiviraj K. Muthumanickam, Katerina Vrotsou, Matthew Cooper, and Jimmy Johansson. Shape grammar extraction for efficient query-by-sketch pattern matching in long time series. In *Proceedings of the 2016 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pages 121–130, 2016.

[71] Michael Correll and Michael Gleicher. The semantics of sketch: Flexibility in visual query systems for time series data. In *Proceedings of the 2016 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pages 131–140, 2016.

[72] Harry Hochheiser and Ben Shneiderman. Dynamic query tools for time series data sets: timebox widgets for interactive exploration. *Information Visualization*, 3(1):1–18, 2004.

[73] Zhe Wang, Nivan Ferreira, Youhao Wei, Aarthy Sankari Bhaskar, and Carlos Scheidegger. Gaussian Cubes: Real-time modeling for visual exploration of large multidimensional datasets. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):681–690, 2017.

[74] Ted Dunning and Otmar Ertl. Computing extremely accurate quantiles using t-digests. Available: `https://github.com/tdunning/t-digest/`, 2014. Accessed on: Jan. 31, 2018.

[75] Wes McKinney. *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. O'Reilly, 2013.

[76] Nick Hornby. *High Fidelity*. Penguin UK, 2000.

[77] Barry J. Faulk. Love and lists in Nick Hornby's High Fidelity. *Cultural Critique*, 66(1):153–176, 2007.

[78] Todd Mostak. An overview of MapD (massively parallel database). In *White paper*. Massachusetts Institute of Technology, 2013.

[79] Zhicheng Liu, Biye Jiang, and Jeffrey Heer. imMens: Real-time visual querying of big data. *Computer Graphics Forum*, 32(3):421–430, 2013.

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

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

[82] Alan Dix and Geoff Ellis. By chance - enhancing interaction with large data sets through statistical sampling. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 167–176, 2002.

[83] Ben Shneiderman. Dynamic queries for visual information seeking. *IEEE Software*, 11(6):70–77, 1994.

[84] Daniel B. Carr, Richard J. Littlefield, Wesley L. Nicholson, and J. S. Littlefield. Scatterplot matrix techniques for large n. *Journal of the American Statistical Association*, 82(398):424–436, 1987.

[85] Peter J. Rousseeuw and Annick M. Leroy. *Robust regression and outlier detection*, volume 589. John Wiley & Sons, 2005.

[86] Niranjan Kamat, Prasanth Jayachandran, Karthik Tunga, and Arnab Nandi. Distributed and interactive cube exploration. In *Proceedings of the 2014 IEEE 30th International Conference on Data Engineering*, pages 472–483, 2014.

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

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

[89] Lisi Chen, Gao Cong, Christian S. Jensen, and Dingming Wu. Spatial keyword query processing: an experimental evaluation. In *Proceedings of the 39th International Conference on Very Large Data Bases*, pages 217–228. VLDB Endowment, 2013.

[90] Subodh Vaid, Christopher B. Jones, Hideo Joho, and Mark Sanderson. Spatio-textual indexing for geographical search on the web. In *Proceedings of the 9th International Conference on Advances in Spatial and Temporal Databases*, pages 218–235, 2005.

[91] Ariel Cary, Ouri Wolfson, and Naphtali Rishe. Efficient and scalable method for processing top-k spatial boolean queries. In *Scientific and Statistical Database Management*, pages 87–95. Springer, 2010.

[92] João B. Rocha-Junior, Orestis Gkorgkas, Simon Jonassen, and Kjetil Nørvåg. Efficient processing of top-k spatial keyword queries. In *Advances in Spatial and Temporal Databases*, pages 205–222. Springer, 2011.

[93] Dongxiang Zhang, Kian-Lee Tan, and Anthony KH Tung. Scalable top-k spatial keyword search. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 359–370, 2013.

[94] Dong Xin, Jiawei Han, Hong Cheng, and Xiaolei Li. Answering top-k queries with multi-dimensional selections: The ranking cube approach. In *Proceedings of the 32nd International Conference on Very Large Data Bases*, pages 463–474. VLDB Endowment, 2006.

[95] Tianyi Wu, Dong Xin, and Jiawei Han. Arcube: supporting ranking aggregate queries in partially materialized data cubes. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of data*, pages 79–92, 2008.

[96] Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys*, 40(4):11:1–11:58, 2008.

[97] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, 66(4):614–656, 2003.

[98] Michal Shmueli-Scheuer, Chen Li, Yosi Mass, Haggai Roitman, Ralf Schenkel, and Gerhard Weikum. Best-effort top-k query processing under budgetary constraints. In *Proceedings of the 2009 IEEE 25th International Conference on Data Engineering*, pages 928–939, 2009.

[99] Nivan Ferreira, Lauro Lins, Daniel Fink, Steve Kelling, Chris Wood, Juliana Freire, and Claudio T. Silva. BirdVis: Visualizing and understanding bird populations. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2374–2383, 2011.

[100] Jo Wood, Jason Dykes, Aidan Slingsby, and Keith Clarke. Interactive visual exploration of a large spatio-temporal dataset: reflections on a geovisualization mashup. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1176–1183, 2007.

[101] Conglei Shi, Weiwei Cui, Shixia Liu, Panpan Xu, Wei Chen, and Huamin Qu. RankExplorer: Visualization of ranking changes in large time series data. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2669–2678, 2012.

[102] Samuel Gratzl, Alexander Lex, Nils Gehlenborg, Hanspeter Pfister, and Marc Streit. Lineup: Visual analysis of multi-attribute rankings. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2277–2286, 2013.

[103] Charles Perin, Romain Vuillemot, and Jean-Daniel Fekete. A table!: Improving temporal navigation in soccer ranking tables. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 887–896, 2014.

[104] Wikimedia downloads. Available: `https://dumps.wikimedia.org/`. Accessed on: Mar. 31, 2015.

[105] Georgios Gousios. The GHTorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 233–236, 2013.

[106] NYC Department of City Planning. East midtown environmental impact statement. Available: `https://www1.nyc.gov/assets/planning/download/pdf/applicants/env-review/gem/05_feis.pdf`. Accessed on: Oct. 23, 2017.

[107] Prevision Design. Evaluation of new shadow generation from proposed development at 888 Tennessee street per SF planning section 295 standards. Available: `http://sfrecpark.org/wp-content/uploads/Item-5-888-Tennessee-Attachment-D_Shadow-Study-060717-a.pdf`. Accessed on: Oct. 23, 2017.

[108] Elmar Eisemann, Michael Schwarz, Ulf Assarsson, and Michael Wimmer. *Real-Time Shadows.* A.K. Peters, 2011.

[109] Andrew Woo and Pierre Poulin. *Shadow Algorithms Data Miner.* CRC Press, 2012.

[110] Google. Project Sunroof data explorer: a description of methodology and inputs. Available: `https://www.google.com/get/sunroof/data-explorer/data-explorer-methodology.pdf`. Accessed on: Oct. 23, 2017.

[111] Autodesk. Solar access with Insight Revit 2017 plugin. Available: `http://blogs.autodesk.com/insight/solar-access-with-insight-revit-2017-plugin/`. Accessed on: Oct. 23, 2017.

[112] Tassilo Glander and Jürgen Döllner. Abstract representations for interactive visualization of virtual 3D city models. *Computers, Environment and Urban Systems*, 33(5):375–387, 2009.

[113] Wei Zeng, Chi-Wing Fu, S.M. Arisona, A. Erath, and Huamin Qu. Visualizing mobility of public transportation system. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1833–1842, 2014.

[114] Zuchao Wang, Min Lu, Xiaoru Yuan, Junping Zhang, and Huub Van De Wetering. Visual traffic jam analysis based on trajectory data. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2159–2168, 2013.

[115] Gennady Andrienko and Natalia Andrienko. Spatio-temporal aggregation for visual analysis of movements. In *Proceedings of the 2008 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pages 51–58, 2008.

[116] Abdullah Kurkcu, Fabio Miranda, Kaan Ozbay, and Claudio T. Silva. Data visualization tool for monitoring transit operation and performance. In *Proceedings of the 2017 IEEE 5th International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*, pages 598–603, 2017.

[117] Huamin Qu, Wing-Yi Chan, Anbang Xu, Kai-Lun Chung, Kai-Hon Lau, and Ping Guo. Visual analysis of the air pollution problem in Hong Kong. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1408–1415, 2007.

[118] Tuan-Anh Hoang-Vu, Vicki Been, Ingrid Gould Ellen, Max Weselcouch, and Juliana Freire. Towards understanding real-estate ownership in New York City: Opportunities and challenges. In *Proceedings of the International Workshop on Data Science for Macro-Modeling*, pages 1–2, 2014.

[119] GuoDao Sun, RongHua Liang, FuLi Wu, and HuaMin Qu. A web-based visual analytics system for real estate data. *Science China Information Sciences*, 56(5):1–13, 2013.

[120] Jiawan Zhang, E Yanli, Jing Ma, Yahui Zhao, Binghan Xu, Liting Sun, Jinyan Chen, and Xiaoru Yuan. Visual analysis of public utility service problems in a metropolis. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1843–1852, 2014.

[121] Yixian Zheng, Wenchao Wu, Yuanzhe Chen, Huamin Qu, and Lionel M. Ni. Visual analytics in urban computing: An overview. *IEEE Transactions on Big Data*, 2(3):276–296, 2016.

[122] PlaceILive. PlaceILive.com. Available: `http://placeilive.com/`. Accessed on: Oct. 23, 2017.

[123] Transitmix. Transitmix. Available: `http://www.transitmix.net/`. Accessed on: Oct. 23, 2017.

[124] Flux. Flux data sharing platform for building construction and design. Available: `http://flux.io/`. Accessed on: Oct. 23, 2017.

[125] UrbanSim Inc. ViziCities. Available: `http://vizicities.com/`. Accessed on: Oct. 23, 2017.

[126] Kevin Johnston, Jay M Ver Hoef, Konstantin Krivoruchko, and Neil Lucas. *Using ArcGIS geostatistical analyst*. ESRI, 2001.

[127] Thomas Ortner, Johannes Sorger, Harald Steinlechner, Gerd Hesina, Harald Piringer, and Eduard Gröller. Vis-A-Ware: Integrating spatial and non-spatial visualization for visibility-aware urban planning. *IEEE Transactions on Visualization and Computer Graphics*, 23(2):1139–1151, 2017.

[128] Ralph L. Knowles. *Energy and Form; an Ecological Approach to Urban Growth*. MIT Press, 1974.

[129] Ralph L. Knowles. *Sun Rhythm Form*. MIT Press, 1981.

[130] Ralph L. Knowles. The solar envelope: its meaning for energy and buildings. *Energy and Buildings*, 35(1):15–25, 2003.

[131] Paul Littlefair. Passive solar urban design: ensuring the penetration of solar energy into the city. *Renewable and Sustainable Energy Reviews*, 2(3):303–326, 1998.

[132] Paul Littlefair. Daylight, sunlight and solar gain in the urban environment. *Solar Energy*, 70(3):177–185, 2001.

[133] Isaac G. Capeluto and E. Shaviv. On the use of solar volume for determining the urban fabric. *Solar Energy*, 70(3):275–280, 2001.

[134] Raphaël Compagnon. Solar and daylight availability in the urban fabric. *Energy and buildings*, 36(4):321–328, 2004.

[135] Jérôme Henri Kämpf, Marylène Montavon, Josep Bunyesc, Raffaele Bolliger, and Darren Robinson. Optimisation of buildings' solar irradiation availability. *Solar Energy*, 84(4):596–603, 2010.

[136] Paul Richens. Image processing for urban scale environmental modelling. In *Proceedings of the 5th International IBPSA Conference: Building Simulation 97*, pages 163–171, 1997.

[137] Carlo Ratti and Paul Richens. Urban texture analysis with image processing techniques. In *Proceedings of the Eighth International Conference on Computer Aided Architectural Design Futures*, pages 49–64, 1999.

[138] Carlo Ratti and Paul Richens. Raster analysis of urban form. *Environment and Planning B: Planning and Design*, 31(2):297–309, 2004.

[139] Fredrik Lindberg. Towards the use of local governmental 3-d data within urban climatology studies. *Mapping Image Science*, 2:32–37, 2005.

[140] Fredrik Lindberg, Björn Holmer, and Sofia Thorsson. SOLWEIG 1.0–modelling spatial variations of 3D radiant fluxes and mean radiant temperature in complex urban settings. *International Journal of Biometeorology*, 52(7):697–713, 2008.

[141] Fredrik Lindberg and C. S. B. Grimmond. The influence of vegetation and building morphology on shadow patterns and mean radiant temperatures in urban areas: model development and evaluation. *Theoretical and Applied Climatology*, 105(3-4):311–323, 2011.

[142] Fredrik Lindberg, Per Jonsson, Tsuyoshi Honjo, and Dag Wästberg. Solar energy on building envelopes–3D modelling in a 2D environment. *Solar Energy*, 115:369–378, 2015.

[143] S. Freitas, C. Catita, P. Redweik, and M.C. Brito. Modelling solar potential in the urban environment: State-of-the-art review. *Renewable and Sustainable Energy Reviews*, 41:915 – 931, 2015.

[144] C. Catita, P. Redweik, J. Pereira, and M.C. Brito. Extending solar potential analysis in buildings to vertical facades. *Computers & Geosciences*, 66(C):1–12, 2014.

[145] Gunther H. Weber, Hans Johansen, Daniel T. Graves, and Terry J. Ligocki. Simulating urban environments for energy analysis. In *Proceedings of the 2014 Workshop on Visualisation in Environmental Sciences (EnvirVis)*, 2014.

[146] J. Alstan Jakubiec and Christoph F. Reinhart. A method for predicting city-wide electricity gains from photovoltaic panels based on LiDAR and GIS data combined with hourly daysim simulations. *Solar Energy*, 93:127 – 143, 2013.

[147] Fan Zhang, Hanqiu Sun, Leilei Xu, and Lee Kit Lun. Parallel-split shadow maps for large-scale virtual environments. In *Proceedings of the 2006 ACM International Conference on Virtual Reality Continuum and Its Applications*, pages 311–318, 2006.

[148] D. Brandon Lloyd, David Tuft, Sung-eui Yoon, and Dinesh Manocha. Warping and partitioning for low error shadow maps. In *Proceedings of the 17th Eurographics Conference on Rendering Techniques*, pages 215–226, 2006.

[149] Erik Sintorn, Viktor Kämpe, Ola Olsson, and Ulf Assarsson. Compact precomputed voxelized shadows. *ACM Transactions on Graphics*, 33(4):150:1–150:8, 2014.

[150] Viktor Kämpe, Erik Sintorn, Dan Dolonius, and Ulf Assarsson. Fast, memory-efficient construction of voxelized shadows. *IEEE Transactions on Visualization and Computer Graphics*, 22(10):2239–2248, 2016.

[151] Marc Stamminger and George Drettakis. Perspective shadow maps. *ACM Transactions on Graphics*, 21(3):557–562, 2002.

[152] Michael Wimmer, Daniel Scherzer, and Werner Purgathofer. Light space perspective shadow maps. In *Proceedings of the 15th Eurographics Conference on Rendering Techniques*, pages 143–151, 2004.

[153] Erik Sintorn, Elmar Eisemann, and Ulf Assarsson. Sample based visibility for soft shadows using alias-free shadow maps. In *Proceedings of the 19th Eurographics Conference on Rendering*, pages 1285–1292, 2008.

[154] Tom Lokovic and Eric Veach. Deep shadow maps. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pages 385–392, 2000.

[155] Daniel Scherzer, Michael Wimmer, and Werner Purgathofer. A survey of real-time hard shadow mapping methods. *Computer Graphics Forum*, 30(1):169–186, 2011.

[156] Hoshang Kolivand and Mohd Shahrizal Sunar. Survey of shadow volume algorithms in computer graphics. *IETE Technical Review*, 30(1):38–46, 2013.

[157] Erik Sintorn, Viktor Kämpe, Ola Olsson, and Ulf Assarsson. Per-triangle shadow volumes using a view-sample cluster hierarchy. In *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 111–118, 2014.

[158] Peter Djeu, Sean Keely, and Warren Hunt. Accelerating shadow rays using volumetric occluders and modified kd-tree traversal. In *Proceedings of the 2009 Conference on High Performance Graphics*, pages 69–76, 2009.

[159] Javor Kalojanov, Markus Billeter, and Philipp Slusallek. Two-level grids for ray tracing on GPUs. *Computer Graphics Forum*, 30(2):307–314, 2011.

[160] Nicolas Feltman, Minjae Lee, and Kayvon Fatahalian. SRDH: Specializing BVH construction and traversal order using representative shadow ray sets. In *Proceedings of the 2012 Conference on High Performance Graphics*, pages 49–55, 2012.

[161] Jae-Ho Nah and Dinesh Manocha. SATO: Surface area traversal order for shadow ray tracing. *Computer Graphics Forum*, 33(6):167–177, 2014.

[162] Elmar Eisemann and Xavier Décoret. Visibility sampling on GPU and applications. *Computer Graphcis Forum*, 26(3):535–544, 2007.

[163] Randima Fernando. *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*. Pearson Higher Education, 2004.

[164] Wolfgang Heidrich, Stefan Brabec, and Hans-Peter Seidel. *Soft Shadow Maps for Linear Lights*, pages 269–280. 2000.

[165] Lance Williams. Casting curved shadows on curved surfaces. In *Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques*, pages 270–274, 1978.

[166] Tobias Martin and Tiow-Seng Tan. Anti-aliasing and continuity with trapezoidal shadow maps. In *Proceedings of the 15th Eurographics Conference on Rendering Techniques*, pages 153–160, 2004.

[167] Renee Cafaro, David Diamond, Joe Ferrara, and Layla Law-Gisiko. Central Park sunshine task force report. Available: `http://www.cb5.org/cb5/projects/central-park-sunshine-task-force_1/`. Accessed on: Oct. 23, 2017.

[168] Municipal Art Society of NY. Accidental skyline. Available: `http://www.mas.org/ourwork/accidental-skyline/`. Accessed on: Oct. 23, 2017.

[169] Thomas J. Lueck. Hundreds rally against towers at Coliseum site. *New York Times*, October 19, 1987. Available: `http://www.nytimes.com/1987/10/19/nyregion/hundreds-rally-against-towers-at-coliseum-site.html`. Accessed on: Oct. 23, 2017.

[170] NYC MOEC. City environmental quality review: Technical manual. Available: `http://www1.nyc.gov/site/oec/environmental-quality-review/technical-manual.page`. Accessed on: Oct. 23, 2017.

[171] Quoctrung Bui and Jeremy White. Mapping the shadows of New York City: Every building, every block. *New York Times*, December 21, 2016. Available: `http://www.nytimes.com/interactive/2016/12/21/upshot/Mapping-the-Shadows-of-New-York-City.html`. Accessed on: Oct. 23, 2017.

[172] Dragomir Anguelov, Carole Dulong, Daniel Filip, Christian Frueh, Stéphane Lafon, Richard Lyon, Abhijit Ogale, Luc Vincent, and Josh Weaver. Google Street View: Capturing the world at street level. *Computer*, 43(6):32–38, 2010.

[173] Aaron Shapiro. Street-level: Google street view's abstraction by datafication. *New Media & Society*, 2017.

[174] Nikhil Naik, Jade Philipoom, Ramesh Raskar, and César Hidalgo. Streetscore-predicting the perceived safety of one million streetscapes. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 779–785, 2014.

[175] Sean M. Arietta, Alexei A. Efros, Ravi Ramamoorthi, and Maneesh Agrawala. City forensics: Using visual elements to predict non-visual city attributes. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2624–2633, 2014.

[176] Jane Jacobs. The death and life of great american cities. 1961.

[177] Yong Rui, Thomas S. Huang, and Shih-Fu Chang. Image retrieval: Current techniques, promising directions, and open issues. *Journal of Visual Communication and Image Representation*, 10(1):39–62, 1999.

[178] Christian Wengert, Matthijs Douze, and Hervé Jégou. Bag-of-colors for improved image search. In *Proceedings of the 19th ACM International Conference on Multimedia*, pages 1437–1440, 2011.

[179] Xiang-Yang Wang, Bei-Bei Zhang, and Hong-Ying Yang. Content-based image retrieval by integrating color and texture features. *Multimedia Tools and Applications*, 68(3):545–569, 2014.

[180] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 806–813, 2014.

[181] Ming-Ming Cheng, Ziming Zhang, Wen-Yan Lin, and Philip Torr. BING: Binarized normed gradients for objectness estimation at 300fps. In *2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3286–3293, 2014.

[182] Liang Zheng, Shengjin Wang, Lu Tian, Fei He, Ziqiong Liu, and Qi Tian. Query-adaptive late fusion for image search and person re-identification. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1741–1750, 2015.

[183] Wengang Zhou, Houqiang Li, and Qi Tian. Recent advance in content-based image retrieval: A literature survey. *arXiv preprint arXiv:1706.06064*, 2017.

[184] Sergio Ilarri, Eduardo Mena, and Arantza Illarramendi. Location-dependent query processing: Where we are and where we are heading. *ACM Computing Surveys*, 42(3):12:1–12:73, 2010.

[185] Harish Doraiswamy, Eleni Tzirita Zacharatou, Fabio Miranda, Marcos Lage, Anastasia Ailamaki, Cláudio T. Silva, and Juliana Freire. Interactive visual exploration of spatio-temporal urban data sets using urbane. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1693–1696, 2018.

[186] Eleni Tzirita Zacharatou, Harish Doraiswamy, Anastasia Ailamaki, Cláudio T. Silva, and Juliana Freiref. GPU rasterization for real-time spatial aggregation over arbitrary polygons. *Proceedings of the VLDB Endowment*, 11(3):352–365, 2017.

[187] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B. Goldman. Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics*, 28(3):24, 2009.

[188] Christine Guillemot and Olivier Le Meur. Image inpainting : Overview and recent advances. *IEEE Signal Processing Magazine*, 31(1):127–144, 2014.

[189] Zhe Zhu, Hao-Zhi Huang, Zhi-Peng Tan, Kun Xu, and Shi-Min Hu. Faithful completion of images of scenic landmarks using internet images. *IEEE Transactions on Visualization and Computer Graphics*, 22(8):1945–1958, 2016.

[190] Jakub Fišer, Ondřej Jamriška, Michal Lukáč, Eli Shechtman, Paul Asente, Jingwan Lu, and Daniel Sýkora. Stylit: Illumination-guided example-based stylization of 3d renderings. *ACM Transactions on Graphics*, 35(4):92:1–92:11, 2016.

[191] Connelly Barnes and Fang-Lue Zhang. A survey of the state-of-the-art in patch-based synthesis. *Computational Visual Media*, 3(1):3–20, 2017.

[192] StreetSide: Dynamic Street-Level Imagery via Bing Maps. Available: `https://www.microsoft.com/en-us/maps/streetside`. Accessed on: Aug. 10, 2018.

[193] Andrew G. Rundle, Michael D.M. Bader, Catherine A. Richards, Kathryn M. Neckerman, and Julien O. Teitler. Using Google Street View to audit neighborhood environments. *American Journal of Preventive Medicine*, 40(1):94–100, 2011.

[194] Xiaojiang Li, Chuanrong Zhang, Weidong Li, Robert Ricard, Qingyan Meng, and Weixing Zhang. Assessing street-level urban greenery using google street view and a modified green view index. *Urban Forestry & Urban Greening*, 14(3):675–685, 2015.

[195] Nikhil Naik, Scott Duke Kominers, Ramesh Raskar, Edward L. Glaeser, and César A. Hidalgo. Computer vision uncovers predictors of physical urban change. *Proceedings of the National Academy of Sciences*, 114(29):7571–7576, 2017.

[196] Carl Doersch, Saurabh Singh, Abhinav Gupta, Josef Sivic, and Alexei A Efros. What makes paris look like paris? *Communications of the ACM*, 58(12):103–110, 2015.

[197] Roberto Carrasco-Hernandez, Andrew R. D. Smedley, and Ann R. Webb. Using urban canyon geometries obtained from Google Street View for atmospheric studies: Potential applications in the calculation of street level total shortwave irradiances. *Energy and Buildings*, 86:340–348, 2015.

[198] Yonglin Zhang and Rencai Dong. Impacts of street-visible greenery on housing prices: Evidence from a hedonic price model and a massive street view image dataset in beijing. *ISPRS International Journal of Geo-Information*, 7(3), 2018.

[199] Kotaro Hara, Jin Sun, Robert Moore, David Jacobs, and Jon Froehlich. Tohme: detecting curb ramps in Google Street View using crowdsourcing, computer vision, and machine learning. In *Proceedings of the 27th annual ACM Symposium on User Interface Software and Technology*, pages 189–204. ACM, 2014.

[200] Vahid Balali, Armin Ashouri Rad, and Mani Golparvar-Fard. Detection, classification, and mapping of us traffic signs using google street view images for roadway inventory management. *Visualization in Engineering*, 3(1):15, 2015.

[201] Christian Lander, Frederik Wiehr, Nico Herbig, Antonio Krüger, and Markus Löchtefeld. Inferring landmarks for pedestrian navigation from mobile eye-tracking data and Google Street View. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, pages 2721–2729, 2017.

[202] Qiaomu Shen, Wei Zeng, Yu Ye, Stefan Müller Arisona, Simon Schubiger, Remo Burkhard, and Huamin Qu. Streetvizor: Visual exploration of human-scale urban forms based on street views. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):1004–1013, 2018.

[203] Ken Sakurada, Daiki Tetsuka, and Takayuki Okatani. Temporal city modeling using street level imagery. *Computer Vision and Image Understanding*, 157:55–71, 2017.

[204] City of New York. 311. Available: `https://www1.nyc.gov/311/`. Accessed on: Oct. 23, 2017.

[205] Johannes Kopf, Billy Chen, Richard Szeliski, and Michael Cohen. Street Slide: browsing street level imagery. In *ACM Transactions on Graphics*, volume 29, pages 96:1–96:8, 2010.

[206] Carlos A. Vanegas, Ignacio Garcia-Dorado, Daniel G. Aliaga, Bedrich Benes, and Paul Waddell. Inverse design of urban procedural models. *ACM Transactions on Graphics*, 31(6):168, 2012.

[207] Harish Doraiswamy, Nivan Ferreira, Marcos Lage, Huy Vo, Luc Wilson, Heidi Werner, Muchan Park, and Cláudio Silva. Topology-based catalogue exploration framework for identifying view-enhanced tower designs. *ACM Transactions on Graphics*, 34(6):230, 2015.