SIGCHI Conference Proceedings Format

Chanwut Kittivorawong
UC Berkeley, USA
chanwutk@berkeley.edu

Shadaj Laddad
UC Berkeley, USA
shadaj@berkeley.edu

Andrew Lenz
UC Berkeley, USA
andrew.lenz@berkeley.edu

Amy Lu
UC Berkeley, USA
amyxlu@berkeley.edu

ABSTRACT

Geospatial-video databases help data scientists store their video data efficiently, but many of them do not provide intuitive query languages for the data scientists to retrieve the data back. Databases with intuitive query languages provide data abstractions that hide their internal data representations. These data abstractions limit the number of operations that data scientists can perform on their data.

In this paper, we discuss our need-finding interviews with potential users of geospatial-video data analysis tools. From the interviews, the existing geospatial-video query language cannot express queries that refer to videos by frames. As a response, we present improvements to the expressiveness of the existing geospatial-video data analysis tool. We focus on its data abstraction and query language. Our new data abstraction additionally includes a video frame data-type. Our new query language provides operations for users to reason about video frames in their data.

Author Keywords

Authors' choice; of terms; separated; by semicolons; include commas, within terms only; this section is required.

CCS Concepts

•**Human-centered computing** → **Human computer interaction (HCI);** *Haptic devices;* User studies; Please use the 2012 Classifiers and see this link to embed them in the text: https://dl.acm.org/ccs/ccs_flat.cfm

INTRODUCTION

[Mick: todo - make sure to address motivation as well
- I (Andrew) gave a rough answer to some of the questions posed in the guidelines, but they could probably use another look
- Why is this area important/why should a reader care about the work you did?

We want to make it easier to search through massive video datasets, especially given how important the research that users do on these datasets is (TODO: Fix this sentence!!!) Data journalists often have large banks of video data, but not nearly enough time to look through this data, which can make it hard to find evidence of police misconduct. Machine learning researchers who work with autonomous vehicles often want to find the most tricky scenarios in order to train the vehicles to maneuver such situations. [further explanation needed]
- What will be the outcome if your project is successful?
We hope to create a tool that will allow programmers of any experience level to easily explore large video databases. - Why hasn't this topic been studied before, or why haven't prior studies answered the questions you're answering?
One issue with developing this tool is that in order to use it, there must be an annotated dataset that contains information such as the geographical location of objects and labels of the elements in the video. However, some of these concerns could be amended by using a computer vision model to label the video, so we will not address that in this paper.
- Why hasn't this problem been solved before, or why are existing solutions insufficient? –> seems better suited for Related Work]

RELATED WORK

[Mick: todo]
[Todo: shadaj]
[Todo: andrew: talk about Fructure for Racket as inspiration for the UI design]

INTERVIEWS

[Mick: todo] [Todo: andrew?]

DATA MODEL

[Mick: todo] [Todo: to focus: the added idea of exploring video data by frames instead if by instances like in apperception] [Todo: shadaj]

DATASET

For our prototype, we use the nuScenes dataset [], a collection of image, lidar, and radar to facilitate autonomous driving research. Data is collected from 15 hours of driving data in Boston and Singapore, and key frames are sampled at a rate of 2Hz for labelling by human annotators, resulting in a rich collection of information suitable for various downstream

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

needs in autonomous driving research. We choose this dataset as its schema is conducive to our aim of.

The entities which the users must interact with are abstracted into *annotation, instance, frame*, and *video*. Specifically, by specifying a string-identifier of a scene, the user can returns `Video` consisting of a list of `Frames`. Each `Frame` consists of several `instances` accompanied by `annotations`. This simple interface abstracts away the need to interact with the native nuScenes schema and offers easy access to information which is likely to be used, and furthermore provides a coarse-to-finegrained view of the data, which arose during the user need-finding studies.

LANGUAGE

[Todo: to focus:
- by-frames operations: as a response to Lisa's use case where we need to scope the output video to the only interesting parts of each videos.
- by-frames <–> by-instances: as a response to Yousef's comments to allow the language to express more queries]

To address the pain points we noticed in our user interviews, we chose to design a new language for describing queries over geospatial video data as a domain-specific language embedded in Python. Instead of building a new language from scratch, which would require designing a new syntax and editor integrations that users would have to learn about, we are able to inherit the large ecosystem of existing tooling around Python. Furthermore, some potential users of our system are already familiar with Python, which makes it further easier to integrate into existing pipelines.

Design Guidelines

There are three key factors constraining the design of our language: incremental query creation, support for using the language through a graphical user interface, and performant query execution. Before we dive into how we design the language around these constraints, let us walk through how we derived the factors based on our user-studies and their consequences on the space of language designs.

Incremental Queries

Across the interviews, a key pattern that stuck out was the process by which the participants would identify portions of video that were of interest—either by manual scrubbing through video files or the creation of queries. Rather than define the end-to-end query, the participants would start with a coarse grained, looking for portions of the video that are likely to have the information they are looking for. Across these sections, the participants would then refine their search/query by playing back the video in real-time or querying individual frames in more detail.

Our insight from this observation was that users of video query languages will want to develop the query incrementally, where they can start with a query that identifies sections of the video with general patterns related to their specific goal (such as the presence of multiple cars in the same frame) and iteratively refine it to identify the specific scenario (such as cars moving towards each other) while receiving feedback from the query engine.

Graphical User Interface

TODO

Performant Query Execution

TODO

Language Design

TODO

Prototype Implementation

TODO?

EXTENSIONS

While we expect that the language will be sufficiently verbose to construct most types of queries, we also recognize that some of our users may not have the time, desire, or programming expertise to implement the functions that they may want to use to filter the videos. Therefore, we built out some of the possible helper functions that could be useful for filtering. These functions expect to be called within filter and thus all return boolean values. The ones provided below should take in the result of the sliding operation, as they intend to compare information about the instance across adjacent frames in order to determine something about the instance's movement.

Utility functions for filtering:
(Note: `iX` and `fY` (where `X` and `Y` are whole numbers) will refer to Instances and Frames, respectively.)

- `move_away(i1, i2, f1, f2)`:
  This function calculates the distance between Instance `i1` and Instance `i2` in Frame `f1` and Frame `f2`, then compares these distances. If the distance between the instances in Frame `f2` is greater than the difference between the instances in Frame `f2`, then we consider the instances to be moving away from one another.
- `accelerating(i, f1, f2, f3)`:
  This function uses the location of Instance `i` to determine if it has accelerated between Frame `f1` and Frame `f2`, and Frame `f2` and Frame `f3`. To determine this, it compares the difference in location between both pairs of frames: if the difference between the locations in Frame `f2` and Frame `f3` is greater than the difference between the locations in Frame `f1` and Frame `f2`, we consider Instance `i` to have accelerated.
- `decelerating(i, f1, f2, f3)`:
  This function uses the location of Instance `i` to determine if it has decelerated between Frame `f1` and Frame `f2`, and Frame `f2` and Frame `f3`. To determine this, it compares the difference in location between both pairs of frames: if the difference between the locations in Frame `f2` and Frame `f3` is less than the difference between the locations in Frame `f1` and Frame `f2`, we consider Instance `i` to have decelerated.
- `stopped(i, f1, f2, tol: float)`:
  This function uses the location of Instance `i` to determine if it has stopped from Frame `f1` to Frame `f2`. To determine

this, it compares the difference in location between the two frames to the tolerance `tol`: if the difference is greater than `tol`, we consider the instance to have moved; otherwise, it has stopped.

USER INTERFACE

Although we did not implement a user interface for [Todo: TOOL NAME], we were able to prototype some designs for such an interface. Based on these explorations, we propose a block-based projectional editor interface to provide the most convenience and accessibility for programmers, particularly those with less experience who we expect to use this tool. To facilitate this, we would allow users to fill the holes in skeleton queries by selecting entries from a drop-down menu which would also have a search bar; this will provide suggestions to users who want to explore the dataset, but also allow users who know what want to see to search for what they are looking for.

As the output of a query in this language would be a series of clips or videos taken from the overall collection of videos, we envision the interface as having some region in which the output of the queries are displayed. However, this creates some issues, as video data is large. To amend this, we would [Todo: how will we display videos? ask Shadaj?]

Mapping the Language to the Interface

[Todo: to focus:
- how we map the programming language to the user interface. For example, how do we present our data model visually in the graphical user tool.
- how do the queries that we can construct in the graphical user tool reflect the queries that we can construct with our query language.
- We do not have to try to cover all the cases that the query language can produce, but we should try to capture the one that are important. For example, how to query by frames, how to query by instances, how to represent the lambda function, how to represent the custom function.]
[Todo: Andrew]
[Todo: Mick]

CONCLUSION

[Mick: todo]

ACKNOWLEDGMENTS

[Mick: todo]

REFERENCES