VisualWorldDB: A DBMS for the Visual World

Brandon Haynes[1], Maureen Daum[1], Amrita Mazumdar[1],
Magdalena Balazinska[1], Alvin Cheung[2], and Luis Ceze[1]

[1]Paul G. Allen School for Computer Science & Engineering, University of Washington
{bhaynes, mdaum, amrita, magda, luisceze}@cs.washington.edu

[2]Department of Electrical Engineering and Computer Sciences, University of California, Berkeley
akcheung@cs.berkeley.edu

ABSTRACT

Many recent video applications—including autonomous driving, traffic monitoring, drone analytics, large-scale surveillance networks, and virtual reality—require reasoning about, combining, and operating over many video streams, each with distinct position and orientation. However, modern video data management systems are largely designed to process individual streams of video data as if they were independent and unrelated.

In this paper, we present VisualWorldDB, a vision and an initial architecture for a new type of database management system optimized for multi-video applications. VisualWorldDB ingests video data from many perspectives and makes them queryable as a *single multidimensional visual object*. It incorporates new techniques for optimizing, executing, and storing multi-perspective video data. Our preliminary results suggest that this approach allows for faster queries and lower storage costs, improving the state of the art for applications that operate over this type of video data.

1. INTRODUCTION

Video data management has recently re-emerged as an active research area due to the proliferation of video cameras, advances in machine learning and graphics hardware, and the emergence of new video-oriented applications (e.g., virtual reality, autonomous driving, surveillance networks, adaptive streaming). New video-oriented data management systems and frameworks (VDBMSs) are specially designed to support these applications (e.g., machine learning [15, 19, 20], analytics [2, 21, 23, 28], virtual and augmented reality [10, 12, 34]).

Today's VDBMSs, however, assume each video stream is *independent*, thus requiring users to be responsible for combining, aligning, downsampling, overlaying, and intermixing sets of videos. At the same time, our world has become filled with correlated cameras that capture what is happening around us from many diverse perspectives and overlapping fields of view. While content produced by an individual camera is valuable, that content would become *richer and much more useful* with the ability to easily combine and reason about streams from *groups* of cameras. For example, consider multiple traffic and vehicle-mounted cameras that capture the scene of an automobile accident. While each camera captures one perspective and potentially the scene of the accident, the combined camera output can help better reconstruct the sequence of events.

Additionally, current VDBMSs do not expose high-level support for, or reasoning about, a video source's dynamic spatiotemporal position or orientation, and has little support for reasoning about visual overlap and field of view between cameras. They also require tedious management of low-level details such as compression, resolution, or frame rate. Finally, many systems fail to support intermixing and querying over the higher-level properties required by recent video applications (e.g., the trajectory of or number of passengers in an automobile prior to an accident).

Despite this lack of support, an important class of emerging applications rely on exactly these concepts: spatiotemporal position, orientation, field of view, and inter-camera overlap. These types of applications are abundant and include diverse areas such as drone analytics [35, 36], civil engineering [3, 9], surveillance [1, 7, 37], autonomous driving, intelligent infrastructure [41], retail analytics [31], and unstructured stitching [27]. These applications, which we refer to as *visual world applications (VWAs)*, all require reasoning about or operations over one or more of these additional concepts, and are thus distinguished from previous-generation video-oriented applications that perform modifications to groups of location- and orientation-agnostic, independent video streams. Implementing VWAs on current VDBMSs requires overcoming conceptual mismatches and technical challenges that lead to brittle implementations, intermixing of application logic with data plumbing, and failure to exploit current and future optimizations.

In this paper, we present a vision and an initial design for a new type of database management system optimized for VWAs. Our system, VisualWorldDB, ingests video data from diverse sources and *makes them queryable as one multidimensional visual object*. The user can then ask rich queries about the content of this visual object.

A major challenge addressed by VisualWorldDB is the data model to expose. Users need a model that captures a world of video streams collected by many cameras, each located in a different location and pointed in a specific direction. VisualWorldDB exposes a data model that consists of a visual world filled with *cameras* and *pixels* from video data and *objects* specified by users or derived from video data. It enables queries over these entities, including support for higher-level semantic objects (e.g., generated via object recognition algorithms) and properties such as trajectories and orientation.

In addition to the technical challenges that hinder *implementing* VWAs, *optimizing* and *executing* queries in VDBMSs is also a substantial challenge. The need to reason about position and orientation often requires joining together

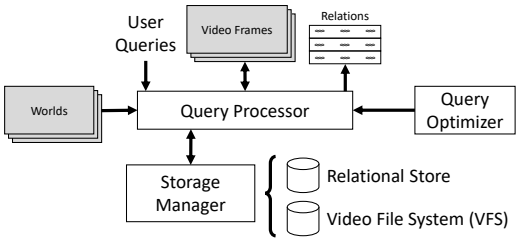
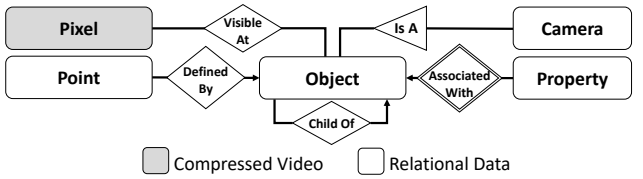


Figure 1: Proposed VisualWorldDB architecture.



Object	$(\underline{oid}, otype, parent)$	Point	$(\underline{pid}, oid, x, y, z, t, ptype)$
Pixel	$(\underline{xid}, x, y, z, t, color)$	Property	$(\underline{oid, key}, value)$
VisibleAt	$(\underline{xid, oid})$		

Figure 2: Entities, relationships, and schema for the VisualWorldDB data model. Objects are defined by points, which define their polyhedral shape, orientation, and trajectory. Objects are also associated with pixels. They may be composed hierarchically and associated with properties.

overlapping video streams (e.g., two traffic cameras that can see the same intersection), resulting in higher complexity and opportunities for optimization. Our prototype deploys new optimization and execution techniques to address these challenges by selecting sets of cameras relevant to a query, and, where possible, reusing results between them rather than reapplying expensive machine learning and computer vision algorithms to recognize objects and infer information.

Finally, optimizations associated with the efficient *storage and retrieval* of physical video data remains underexplored. For example, virtually all video processing systems treat video encoding and decoding algorithms as expensive black boxes, and as a result spend a disproportionate time performing (de)compression relative to useful work. Our storage manager and video file system reduces redundancy in overlapping video data by *jointly compressing* overlapping videos and operates directly on a video's compressed representation.

In summary, we make the following contributions:

- We propose VisualWorldDB, a vision for a data management system for visual world applications (Section 2). VisualWorldDB includes a data model for expressing queries over collections of video as a unified multidimensional object (Section 3).
- We describe and show preliminary results for VisualWorldDB's storage manager, which pushes the state of the art in video storage and retrieval (Section 4).
- We propose query execution and optimization techniques relevant to visual world applications (Section 5).

2. VISUALWORLDDB OVERVIEW

We propose to address the mentioned challenges with the design of VisualWorldDB, as illustrated in Figure 1. To use VisualWorldDB, users begin by creating a *world*, which is an object analogous to a database in a relational system. Each world is a container for interrelated video and higher-level semantic information such as objects and trajectories. To populate the world, users ingest video data and specify the location and orientation of each data source's capturing position.

Using built-in or user-specified computer vision and machine learning algorithms, users next fill the world with *objects* based on the video data and other domain knowledge. Each object is a contextually meaningful entity such as a classification (e.g., an automobile), a region of interest (e.g., a traffic intersection), or an abstraction (e.g., blind spots on an autonomous vehicle). A user fills the world with objects lazily and incrementally, and only materializes the objects that are needed by an application. Each object is associated with a polyhedral shape and optional trajectories and orientation.

Having created and populated a world, users submit declarative queries to VisualWorldDB's query processor. Query answers may be in the form of video data associated with objects (e.g., "find all video of all automobiles exceeding n kilometers per hour") or relational data (e.g., "how many automobiles transited an intersection per hour?"). We provide concrete examples in Section 3.

Upon receiving a query, the query optimizer constructs an execution plan from a set of physical operators and identifies a minimal set of relevant video sources to answer it. The optimizer also reduces the cost of applying machine learning and computer vision algorithms to cameras with overlapping field of view by applying these algorithms to the overlapping region once, intelligently reusing the results for other perspectives (see Section 5).

Similar to prior work [30], VisualWorldDB utilizes a hybrid storage manager that leverages both a relational and video store. As we discuss in Section 4, the video store exploits its knowledge of the 3D spatial configuration of cameras to jointly compress video data, which vastly decreases storage costs. Additionally, it reorganizes compressed data to prioritize frequently-queried pixels and greatly increase the performance of common video operations (e.g., cropping, downsampling).

3. DATA MODEL

Unlike prior VDBMSs with two-dimensional APIs that target video frames, VisualWorldDB exposes a data model that targets applications that reason about multiple video streams in a 3D space. The key idea behind our data model is to directly expose worlds created from overlapping video streams, objects within those videos, and the different perspectives captured by specific videos. Our model is inspired by real-time simulation engines (e.g., [8]) and captures data found in a real or virtual world by organizing and relating video *pixels*; *objects* with shape, orientation, and trajectory defined by *points*; and *cameras* within that world. These entities, their relationships, and the corresponding database schema are shown in Figure 2.

At a high level, a user creates a world and ingests one or more videos. Each imported video is associated with a distinct *camera*, which we treat as a special type of object. If a video has associated depth information, then this data is used to populate a *pixel* relation that stores the spatial (x, y, z) and temporal (t) coordinates of the video's pixels. Otherwise, VisualWorldDB applies a default depth-generation algorithm described by Casser et al. [6] and then populates the relation using estimated depth.

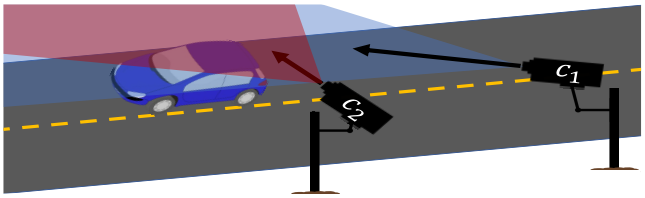


Figure 3: A world consisting of two cameras (c_1 and c_2) respectively positioned at (x_1, y_1, z_1) and (x_2, y_2, z_2) and oriented at $(\theta_1, \phi_1, \rho_1)$ and $(\theta_2, \phi_2, \rho_2)$. Note that the cameras have different positions and orientations and the automobile is visible to both.

Importantly, in either case, pixel data and depth are maintained as compressed video data and only logically exposed as a relation for query authoring (see Section 4.1).

After importing videos, the world contains camera objects and pixels but is devoid of other more interesting objects (and points). A user next populates the world with meaningful objects, usually by inserting the results of an object recognition function. Each such inserted object has a polyhedral shape defined by one or more associated *points*. For example, camera c_1 has a single point at $(pid\colon p_i, oid\colon c_1, x\colon x_1, ..., ptype\colon position)$ that indicates where it is located. Objects may be associated with other types of points as well, including user-defined point types. For example, a camera's viewing direction is given by a special orientation point (e.g., camera c_1 has an orientation point $(pid\colon p_j, oid\colon c_1, x\colon x_1 + \sin(\theta_1), ..., ptype\colon orientation)$). Additionally, each object is associated with the pixels that produced it through the *VisibleAt* relationship; for example, when an "automobile" is inferred by an object recognition algorithm, the "automobile" object is associated with the pixels that were recognized to be a vehicle. Finally, objects can also have a trajectory that is described by the spatial evolution of its points over time.

Each object has a type (e.g., "camera", "automobile", or "cat"). Depending on their type, objects may be visible to cameras or represent invisible semantic annotations (e.g., a convex hull delineating an accident-prone intersection). Finally, objects may be hierarchically composed, e.g., a user might query for the driver inside a particular vehicle or a person's left arm.

All objects can be associated with an arbitrary number of key/value properties. In addition to user-defined properties, VisualWorldDB automatically maintains required properties that includes camera's interpolation method and field of view, object trajectory interpolation, and the units used in the world coordinate system (specified during world creation).

Users may extract videos from cameras—at any resolution, frame rate, and using any compression method—and VisualWorldDB automatically identifies the most efficient way to generate that video and deliver it to the user. Additionally, users may leverage both built-in and user-defined functions that automatically convert (potentially compressed) video into a format compatible with the pixel relation. We describe concrete examples of video import and export below.

To illustrate, we show example queries over the VisualWorldDB data model using SQL syntax in Table 1. However, VisualWorldDB will also expose a functional interface (similar to VRQL [12]) that provides equivalent functionality.

Table 1: Sample VisualWorldDB queries

#	Query
Q1	Create world & populate with videos

```
CREATE WORLD W(units=metric)
INSERT INTO W.Objects (oid, type, parent)
  VALUES ("c1", "camera", NULL)
INSERT INTO W.Points VALUES ("p1", "c1", x1, ..., NULL, "pos")
INSERT INTO W.Points VALUES ("p2", "c1", x1 + sin(θ1), ...,
                             NULL, "orientation")
// Procedure that populates Pixel and VisibleAt relations
EXECUTE W.AddCameraVideoData("c1", "camera1.mp4")

... // Repeat process for c2
```

Q2	Perform recognition on 448 × 448 raw video extracted from c1

```
INSERT INTO W.Objects
  SELECT YOLO(SELECT * FROM W.Pixels
    NATURAL JOIN W.VisibleTo WHERE oid = "c1"
    AS RAWVIDEO WITH RESOLUTION(448, 448))
```

Q3	Manually create an automobile with a trajectory

```
INSERT INTO W.Objects (oid, otype, parent)
  VALUES ("a1", "auto", NULL)
// For simplicity, assume only one automobile in W.Objects
// Add points created in Q2 to the new object
INSERT INTO W.Points SELECT "a1", x, y, z, t, ptype
  FROM W.Points WHERE oid IN (SELECT oid FROM W.Objects
    WHERE otype = "auto")
```

Q4	Get a 128x128 video of automobile a1 from c2

```
SELECT * FROM W.Objects
NATURAL JOIN W.VisibleTo
NATURAL JOIN W.Pixels
WHERE Objects.oid = "c2" AND
      t IN (SELECT t from W.Points P
            WHERE P.oid = "a1")
WITH RESOLUTION(128, 128)
```

Q5	Add a new property to an object

```
INSERT INTO W.Properties VALUES ("a1", "plate", "123ABCD")
```

Our example, illustrated in Figure 3, consists of two overlapping RGB-D cameras (which capture both color and depth information) recording a roadway. A developer begins by issuing Q1 in Table 1 to create a new world containing the two cameras in Figure 3 (`c1` and `c2`). She then adds two points to each new camera: a "position" point indicating the camera's location, and an "orientation" point in the direction of the camera's orientation relative to its position. She finally ingests the video data associated with each camera.

Because the two videos were recorded using RBG-D cameras, they both contain depth information. VisualWorldDB uses that information rather than performing automatic inference. After the query completes, VisualWorldDB contains a world named `W`, two cameras in `W.Camera`, and tuples in `W.Pixels` for each pixel in the source videos. Each pixel is spatiotemporally positioned using time and depth information from its source video. However, despite logically exposing pixels in a relational manner, VisualWorldDB continues to physically store all video data in an efficient compressed manner that we describe in Section 4.1.

Next, the developer identifies semantically relevant objects (i.e., automobiles) visible in the world. To do so, she runs Q2, which performs object recognition using the YOLO built-in function [29] applied to video data from `c1` (we will discuss optimizations related to object recognition on `c2` in Section 5). The resulting objects are automatically projected into three dimensions using the minimum and maximum depth associated with the classified pixels. Importantly, as is common in image-based deep learning algorithms, the neural network used by YOLO requires a *downsampled* frame resolution (448 × 448 for YOLOv2 [29]). In the example, the user elects to request this resolution explicitly and, as we

describe in Sections 4 and 5, VisualWorldDB's optimization, execution, and storage components coordinate to perform this decompression and downsampling efficiently.

Most object *recognition* algorithms identify objects in a single image or frame and do not perform *tracking* of an object across multiple frames. In the previous example, if 1000 frames contained one automobile, `W.Objects` would contain 1000 tuples. In real-world scenarios, manually reconciling the trajectory of many automobiles across frames is unmanageable, and users would employ an object tracking algorithm such as the built-in GOTURN function [14] to automatically reconcile objects and produce trajectories for many moving objects. Due to space constraints we show a simplified example in Q3 for a world that contains a single automobile.

Once relevant objects have been identified and reconciled, the developer is now able to execute complex queries over the world that would be challenging to express in existing video-oriented systems. Assume an automobile `a1` was identified by the previous query and that the developer wishes to extract video from `c2` when the automobile was visible on `c1`. To do so she executes Q4. Note that in this example, we performed object recognition on camera `c1` but are able to automatically extract correlating video from *a different camera* that is more likely to contain video of the driver. We describe further optimizations related to overlapping cameras in Section 5.

Finally, as mentioned, the developer may also create new properties and query them. For example, Q5 associates a license plate with the suspected automobile `a1` detected previously.

We have carefully designed the VisualWorldDB model to be simple so that users can easily write queries over a real or virtual world. However, this simplicity raises a number of challenges in storage, back-end implementation, and query optimization, which we discuss in the next sections.

4. STORAGE SYSTEM

Current VDBMSs treat video compression as a black box, either storing data as a single compressed file or uncompressed as raw pixels. Instead, our vision is to *decouple* compression decisions from an application and exploit the resulting data independence to improve performance.

As such, we have partitioned video storage into two components: a storage manager (SM) and a video file system (VFS). This architecture allows other video-oriented systems to leverage each component individually; for example, any VDBMS that can read video from a mounted file system may immediately benefit from the optimizations offered by the VFS.

4.1 Video File System

At the lowest layer, the VFS is responsible for video data storage on disk. It exposes a POSIX-compliant file system interface through which applications (including the SM) read and write video data. The VFS optimizes how videos are compressed and laid out on disk: it chooses compression strategies, fragments videos, and reorganizes pixels within frames. Additionally, rather than treating compressed video as an opaque flat file containing frames that must be sequentially decompressed, the VFS operates directly on the compressed representation whenever possible. This strategy differs from existing systems that sequentially (de)compress video data from the file system before performing useful work on the uncompressed representation.

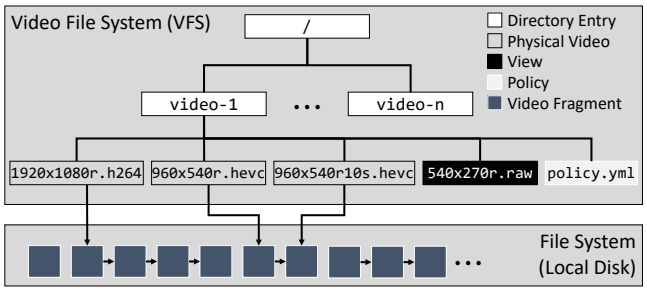


Figure 4: The video file system (VFS) directory structure. Each video is organized under its own directory entry, which contains materializations, views, and a policy file. Materialized views are associated with a sequence of compressed video blocks, which may have redundant physical organization.

As shown in Figure 4, the VFS arranges each video within its own subdirectory. Each subdirectory contains a *policy*, one or more *materializations*, and zero or more *views*. A video's policy determines optimizations that may be applied by the VFS (e.g., favoring performance over quality).

The VFS exposes as API a simple query language with predicates embedded directly in a filename. These predicates include resolution, frame rate, video codec, start and end times, crop locations, and individual frame selection. For example, Q2 in Table 1 requests a 448×448 representation of video from camera c_1, and the SM would read file `/c1/448x448r.raw` to load uncompressed pixels at this resolution.

The VFS caches one or more read-only materializations of a each video, which applications may read efficiently and pin if they expect to frequently do so. A request for any other filename requires the VFS to generate the new view based on one or more materializations, which is returned and possibly stored on disk for future requests.

The VFS additionally employs a battery of optimizations to improve query performance. One example, compressed cropping, allows a user to retrieve cross sections of a frame without decompressing the entire frame. Consider the example in Figure 3, and assume that a user has identified an automobile near the top of camera c_1's field of view. A user might issue a query requesting the uppermost 256 pixels of camera c_1, triggering the SM to request `/c1/256h.h264` from the VFS (the `h` indicates the VFS should crop at the given height). The VFS would apply the compressed crop optimization to decode only the minimal top portion of the video required to answer the query. Our preliminary results suggest that compressed cropping yields more than a $2.5\times$ performance benefit relative to the typical approach of full decompression before cropping.

For space, we defer discussion of other optimizations to a future paper. These include pixel reordering, frequency-domain conversions, and an incremental refinement technique similar to database cracking [17].

4.2 Storage Manager

On top of the VFS, VisualWorldDB's storage manager (SM) is responsible for deciding (i) whether to jointly compress video from overlapping cameras, (ii) what strategy to use when subsequently retrieving video compressed in this manner, and (iii) which video to operate on when answering queries.

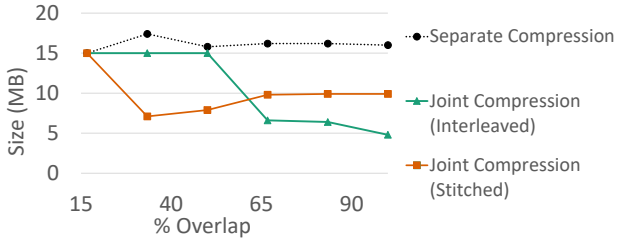


Figure 5: Joint compression performance.

When ingesting or writing video, the first thing the SM must decide is how to physically arrange the data in the VFS. For video produced by physically distant or non-overlapping cameras, the SM simply writes it to a distinct VFS file.

In the case of overlapping videos, the SM exploits its knowledge of each camera's position to automatically combine overlapping camera videos and reduce the storage redundancy in the overlapped regions Concretely, the SM combines video frames from cameras within distance d and with overlapping field of view of at least θ degrees. The SM then writes the joint frames to the VFS, which may apply further optimizations described in Section 4.1. Automatically tuning the d and θ parameters remains as future work.

The SM currently chooses from two joint compression methods. The *interleave* method aligns and alternates rows of pixels in corresponding pairs of video frames and then depends on video compression (applied by the VFS) to minimize the resulting redundancy [32, 38]. The *stitching* method merges pairs of frames into a panorama.

We applied each compression method to fifteen minutes of 4K (3840×2160 pixels) synthetic video data generated using [13] with various degrees of overlap. Figure 5 shows preliminary results in terms of compressed file size for each method. These results suggest that jointly compressing video data reduces compressed video for sizes by up to 60%. We are currently optimizing these methods and exploring additional techniques with different performance trade-offs.

During reads, the SM requests a non-jointly compressed video if available. Otherwise, it requests only relevant video regions from the VFS. For example, if a user executes Q5 from Table 1 after the SM had jointly compressed cameras c_1 and c_2, the SM would request the minimal region that contained data from c_1 and then apply an inverse transformation (e.g., deinterleaving) to recover the original c_1 video. Our preliminary results suggest that joint compression reduces decompression time by an average of 45% when both videos are needed, and adds an average of 30% overhead when only one is required.

5. QUERY EXECUTION & OPTIMIZATION

The basic strategy for executing queries in VisualWorldDB is to translate a query into operations on each video that it touches (and on relational storage), decompress and process each video, then combine results. This approach, however, is inefficient.

Our vision for VisualWorldDB's query optimization and execution focuses on avoiding expensive operations on video data, which has been shown to consume a large amounts of execution time relative to useful work [11, 12]. Specifically, our primary proposed optimization technique involves the *reuse* of results between overlapping videos. Consider the example from Section 3. A user, subsequent to performing object recognition on camera c_1 (i.e., using Q2 from Table 1) might next apply object recognition to camera c_2.

However, note the fields of view between camera c_1 and c_2 have substantial overlap (see Figure 3), and a corresponding redundancy in the objects that exist in the world they view. In this case, VisualWorldDB *omits* applying object recognition on the overlapping region and only applies it to the remainder of the video (plus an optional user-specified margin).

This strategy reduces the work that VisualWorldDB must perform to identify objects, but possibly reduces accuracy due to the omitted video data. For example, an automobile might be occluded on c_1 but not on c_2 due to an intermediating vehicle. To avoid this loss in accuracy, VisualWorldDB additionally refines the overlapping region by automatically including (i) occluded regions identified using pixel depth information, and (ii) regions with a suspected object slightly below the recognition cutoff threshold. This ensures that important overlapping regions visible to c_2 participate in object recognition.

Having identified the smallest region relevant for the application of machine learning or computer vision algorithms, the VisualWorldDB optimizer interacts with the storage manager to decompress the smallest amount of video data as possible. For algorithms applied to overlapping cameras, as in the example above, this might involve requesting video that has been jointly compressed (see Section 4.2) and/or only decompressing cropped subsets of each video frames (e.g., to omit the overlapping areas).

Finally, to improve the performance of ad hoc queries over a frequently-queried object and its associated pixels, we are exploring the construction of spatial indexes over highly-dynamic objects. We envision that the optimizer will leverage this index in conjunction with spatial semijoins [33] to reduce the communication cost between the relational store and video file system (e.g., large joins over pixels associated with many objects).

In addition to the optimizations described above, other possible enhancements could include predicate pushdown and other optimizations described by [21], as well as operators that target heterogeneous hardware [12].

6. RELATED WORK

There has been substantial recent interest in video analytics—especially in the context of applied deep learning—and many previous systems [1, 2, 15, 16, 18, 20, 21, 22, 22, 23, 23, 26, 28, 30, 40, 40, 40] target this domain. Each of these systems target *two-dimensional* video analytics, forcing developers to manually map 3D environments onto a 2D video and reconcile heterogeneous resolutions and frame rates. This results in applications that are difficult to reason about, optimize, maintain, and evolve. VDMS supports rich metadata queries similar to those in VisualWorldDB but only over 2D video frames [30]. Vignette offers improved compression performance by targeting perceptual cues, which are complimentary to optimizations offered by VisualWorldDB [24].

The array-based data models proposed by [4, 5, 25] are similar to the model proposed herein. However, these models generally target scientific workloads and fail to take advantage of video compression optimizations and offer drastically reduced performance for the types of applications that VisualWorldDB targets. Finally, the VisualWorldDB model shares similarities with the model proposed in [12]; however, this model is not optimized to construct and reason about semantically-rich worlds using deep learning and other inference techniques.

Finally, VStore improves performance by offering videos in various formats and resolutions [39] (an approach advocated in prior work [11]). While this interface is similar to that offered by VisualWorldDB's video file system, it requires extensive a priori knowledge of target workloads and is not designed for evolution, ad hoc queries, or online, incremental adaptation.

7. CONCLUSION

In this paper we presented our vision for VisualWorldDB, a new DBMS designed to efficiently support workloads on *visual world applications (VWAs)*, applications that operate on many videos perspective in a real or virtual world. VisualWorldDB comes with a data model, query execution engine, optimizer, storage system, and file system that coordinate to push the state of the art in efficiently executing queries over VWAs. VisualWorldDB allows developers to model high-level objects, quickly identify relationships between those objects and the underlying video data, and extract new perspectives of the world. As our storage system evolves, we will explore the use of other compression-aware techniques such as incremental refinement, new physical transformations and reorganization approaches, and indexing techniques to further improve query performance.

Acknowledgments. This work is supported by the NSF through grants CCF-1703051, CCF-1518703, IIS-1546083, IIS-1651489, OAC-1739419; DARPA award FA8750-16-2-0032; DOE award DE-SC0016260; a Google Faculty Research Award; an award from the University of Washington Reality Lab; Intel Science and Technology Center for Big Data; Intel-NSF CAPA center; Adobe; Huawei; Google; NVIDIA; CRISP, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

8. REFERENCES

[1] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha. Real-time video analytics: The killer app for edge computing. *IEEE Computer*, 50(10):58–67, 2017.

[2] L. Ao, L. Izhikevich, G. M. Voelker, and G. Porter. Sprocket: A serverless video processing framework. In *SoCC*, pages 263–274, 2018.

[3] E. N. Barmpounakis, E. I. Vlahogianni, and J. C. Golias. Unmanned aerial aircraft systems for transportation engineering: Current practice and future challenges. *IJTST*, 5(3):111–122, 2016.

[4] P. Baumann, A. Dehmel, P. Furtado, R. Ritsch, and N. Widmann. The multidimensional database system rasdaman. In *SIGMOD*, pages 575–577, 1998.

[5] P. G. Brown. Overview of SciDB: large scale array storage, processing and analysis. In *SIGMOD*, pages 963–968, 2010.

[6] V. Casser, S. Pirk, R. Mahjourian, and A. Angelova. Depth prediction without the sensors: Leveraging structure for unsupervised learning from monocular videos. In *AAAI*, pages 8001–8008, 2019.

[7] A. Chowdhery and M. Chiang. Model predictive compression for drone video analytics. In *SECON*, pages 19–23, 2018.

[8] Epic Games. Unreal Engine 4. https://www.unrealengine.com, 2019.

[9] S. George, J. Wang, M. Bala, T. Eiszler, P. Pillai, and M. Satyanarayanan. Towards drone-sourced live video analytics for the construction industry. In *HotMobile*, pages 3–8, 2019.

[10] Google Poly. https://poly.google.com.

[11] V. Gupta-Cledat, L. Remis, and C. R. Strong. Addressing the dark side of vision research: Storage. In *HotStorage*, 2017.

[12] B. Haynes, A. Mazumdar, A. Alaghi, M. Balazinska, L. Ceze, and A. Cheung. LightDB: A DBMS for virtual reality video. *PVLDB*, 11(10):1192–1205, 2018.

[13] B. Haynes, A. Mazumdar, M. Balazinska, L. Ceze, and A. Cheung. Visual Road: A video data management benchmark. In *SIGMOD*, pages 972–987, 2019.

[14] D. Held, S. Thrun, and S. Savarese. Learning to track at 100 FPS with deep regression networks. In *ECCV*, pages 749–765, 2016.

[15] K. Hsieh, G. Ananthanarayanan, P. Bodík, S. Venkataraman, P. Bahl, M. Philipose, P. B. Gibbons, and O. Mutlu. Focus: Querying large video datasets with low latency and low cost. In *OSDI*, pages 269–286, 2018.

[16] Q. Huang, P. Ang, P. Knowles, T. Nykiel, I. Tverdokhlib, A. Yajurvedi, P. D. IV, X. Yan, M. Bykov, C. Liang, M. Talwar, A. Mathur, S. Kulkarni, M. Burke, and W. Lloyd. SVE: distributed video processing at Facebook scale. In *SOSP*, pages 87–103, 2017.

[17] S. Idreos, M. L. Kersten, and S. Manegold. Database cracking. In *CIDR*, pages 68–78, 2007.

[18] J. Jiang, G. Ananthanarayanan, P. Bodík, S. Sen, and I. Stoica. Chameleon: scalable adaptation of video analytics. In *SIGCOMM*, pages 253–266, 2018.

[19] D. Kang, P. Bailis, and M. Zaharia. Blazeit: Fast exploratory video queries using neural networks. *CoRR*, abs/1805.01046, 2018.

[20] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia. NoScope: Optimizing deep CNN-based queries over video streams at scale. *PVLDB*, 10(11):1586–1597, 2017.

[21] S. Krishnan, A. Dziedzic, and A. J. Elmore. DeepLens: Towards a visual data management system. In *CIDR*, 2019.

[22] S. Liu, J. Pu, Q. Luo, H. Qu, L. M. Ni, and R. Krishnan. VAIT: A visual analytics system for metropolitan transportation. *ITS*, 14(4):1586–1596, 2013.

[23] Y. Lu, A. Chowdhery, and S. Kandula. Optasia: A relational platform for efficient large-scale video analytics. In *SoCC*, pages 57–70, 2016.

[24] A. Mazumdar, B. Haynes, M. Balazinska, L. Ceze, A. Cheung, and M. Oskin. Perceptual compression for video storage and processing systems. In *SoCC*, pages 179–192, 2019.

[25] S. Papadopoulos, K. Datta, S. Madden, and T. G. Mattson. The TileDB array data storage manager. *PVLDB*, 10(4):349–360, 2016.

[26] Y. Peng, H. Ye, Y. Lin, Y. Bao, Z. Zhao, H. Qiu, Y. Lu, L. Wang, and Y. Zheng. Large-scale video classification with elastic streaming sequential data processing system. In *LSVC*, 2017.

[27] F. Perazzi, A. Sorkine-Hornung, H. Zimmer, P. Kaufmann, O. Wang, S. Watson, and M. H. Gross. Panoramic video from unstructured camera arrays. *Comput. Graph. Forum*, 34(2):57–68, 2015.

[28] A. Poms, W. Crichton, P. Hanrahan, and K. Fatahalian. Scanner: efficient video analysis at scale. *TOG*, 37(4):138:1–138:13, 2018.

[29] J. Redmon and A. Farhadi. YOLO9000: better, faster, stronger. In *CVPR*, pages 6517–6525, 2017.

[30] L. Remis, V. Gupta-Cledat, C. R. Strong, and R. Altarawneh. VDMS: an efficient big-visual-data access for machine learning workloads. *CoRR*, abs/1810.11832, 2018.

[31] A. W. Senior, L. M. Brown, A. Hampapur, C. Shu, Y. Zhai, R. S. Feris, Y. Tian, S. Borger, and C. R. Carlson. Video analytics for retail. In *AVSS*, pages 423–428, 2007.

[32] G. J. Sullivan, J. Ohm, W. Han, and T. Wiegand. Overview of the high efficiency video coding (HEVC) standard. *TCSVT*, 22(12):1649–1668, 2012.

[33] K. Tan, B. C. Ooi, and D. J. Abel. Exploiting spatial indexes for semijoin-based join processing in distributed spatial databases. *TKDE*, 12(6):920–937, 2000.

[34] Google VR View. //developers.google.com/vr/concepts/vrview.

[35] J. Wang, Z. Feng, Z. Chen, S. George, M. Bala, P. Pillai, S. Yang, and M. Satyanarayanan. Bandwidth-efficient live video analytics for drones via edge computing. In *SEC*, pages 159–173, 2018.

[36] X. Wang, A. Chowdhery, and M. Chiang. SkyEyes: adaptive video streaming from UAVs. In *HotWireless*, pages 2–6, 2016.

[37] X. Wang, A. Chowdhery, and M. Chiang. Networked drone cameras for sports streaming. In *ICDCS*, pages 308–318, 2017.

[38] T. Wiegand, G. J. Sullivan, G. Bjntegaard, and A. Luthra. Overview of the H.264/AVC video coding standard. *TCSVT*, 13(7):560–576, 2003.

[39] T. Xu, L. M. Botelho, and F. X. Lin. VStore: A data store for analytics on large videos. In *EuroSys*, pages 16:1–16:17, 2019.

[40] H. Zhang, G. Ananthanarayanan, P. Bodík, M. Philipose, P. Bahl, and M. J. Freedman. Live video analytics at scale with approximation and delay-tolerance. In *NSDI*, pages 377–392, 2017.

[41] J. Zhou, R. Q. Hu, and Y. Qian. A scalable vehicular network architecture for traffic information sharing. *J-SAC*, 31(9-Supplement):85–93, 2013.