# Modeling Illumination Data with Flying Light Specks

Hamed Alimohammadzadeh
University of Southern California
Los Angeles, California, USA

Daryon Mehraban
Corona del Mar High School
Newport Beach, California, USA

Shahram Ghandeharizadeh
University of Southern California
Los Angeles, California, USA

## ABSTRACT

A Flying Light Speck, FLS, is a miniature sized drone configured with light sources. Swarms of FLSs will illuminate an object in a 3D volume, an FLS display. These illuminations and their data models are the novel contributions of this paper. We introduce a conceptual model of drone flight paths to render static, slide, and motion illuminations. We describe a physical implementation of the conceptual model using bag files. We evaluate this implementation using different lossless compression techniques. A key finding is that our bag file implementation is very compact when compared with the original point clouds. While compression reduces the size of a bag file, a combination that includes the use of both internal bag file compression (lz4 with chunks) and Gzip is not necessarily the most compact representation. We open source our software and its point cloud sequence data for use by the scientific community, see https://github.com/flyinglightspeck/FLSbagfile.

## CCS CONCEPTS

• **Information systems** → **Entity relationship models**; **Multimedia information systems**.

## KEYWORDS

3D Display, Point Clouds, Flying Light Specks, Bag file, Flight Paths, Data Representation

**ACM Reference Format:**
Hamed Alimohammadzadeh, Daryon Mehraban, and Shahram Ghandeharizadeh. 2023. Modeling Illumination Data with Flying Light Specks. In *Proceedings of the 14th ACM Multimedia Systems Conference (MMSys '23), June 7–10, 2023, Vancouver, BC, Canada.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3587819.3592544

## 1 INTRODUCTION

An FLS is a miniature sized drone equipped with one or more light sources to generate different colors and textures with adjustable brightness. Synchronized swarms of FLSs will illuminate virtual objects in a pre-specified 3D volume, an FLS display [4–6]. An FLS display may be a cuboid that sits on a table or hangs on a wall, the dashboard of a self-driving vehicle, a room, etc.

An FLS display may render three different types of illuminations. A static illumination renders one point cloud. A slide illumination renders a sequence of point clouds [19]. A motion illumination also renders a sequence of point clouds, however, its point clouds are designed to generate the illusion of motion, e.g., a running man [3], the rose with a falling petal [5], etc.

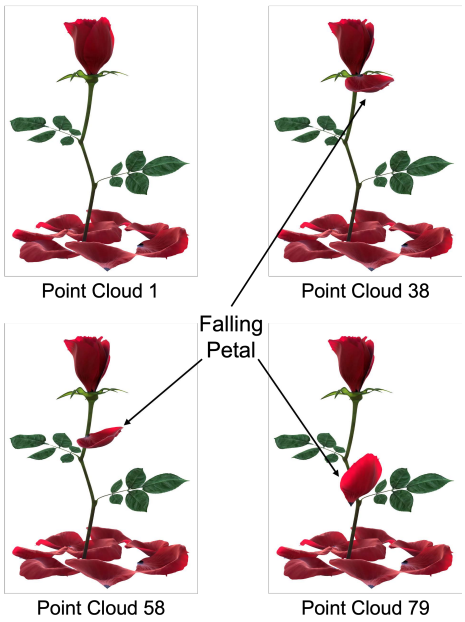


**Figure 1: Four point clouds of the rose with a falling petal.**

A point cloud is the basis of different illuminations. A point dictates the spatial coordinate of an FLS. The point may include attributes such as color, opacity, reflectance, etc. Dynamic content may render a sequence of point clouds at a pre-specified rate to fool the human perception to observe 3D motion. They are used in diverse applications including animation and entertainment, immersive telepresence, cultural heritage archives [7, 16] to name a few. Figure 1 shows four point clouds of a sequence consisting of 115 point clouds. It is an animated rose with a falling petal. Its display time is 4.8 seconds when rendered at 24 point clouds per second, e.g., see https://youtu.be/zaZwAiCsZUU.

A challenge is to compute FLS flight paths that are either collision free [12, 19] or detect possible collisions that are avoided when FLSs render an illumination [5, 8, 9, 20]. Both are time consuming and computationally expensive operations. For sequences that are illuminated repeatedly, one may perform the computation once and store the results in a file. Subsequent illuminations may read the file instead of performing the expensive computation. In addition, these files eliminate the overhead of computing flight paths with simulation studies (e.g., AirSim [17], Gazebo [10]) that investigate alternative architectures for an FLS display [5], lighting designs for FLSs, and algorithms for FLS failure handling and battery charging.


*MMSys '23, June 7–10, 2023, Vancouver, BC, Canada*

ACM ISBN 979-8-4007-0148-1/23/06…$15.00
https://doi.org/10.1145/3587819.3592544

The contributions of this study are several folds. First, it presents a conceptual model of FLS flight paths that render scenes of an animated sequence. Each scene is a motion illumination and corresponds to a sequence of point clouds. While the intra-scene[1] point cloud changes may be smooth, the inter-scene[2] point cloud changes may be drastic. It may be comparable to a slide show switching from one point cloud to the next. With both, the model describes the FLS flight paths and other attributes such as 3D coordinates visited and colors rendered at each coordinate.

Second, we implement a subset of the conceptual model using bag files [13]. Bag files are a customizable format used by the Robot Operating System, ROS [14]. FLSs as drones are a category of robots. Moreover, bag files are extensible. This means both our model and the resulting bag file implementation can be extended with additional features, see Section 6. In addition, software for reading and writing bag files is readily available with different platforms and packages, e.g., MATLAB [11], Python [21], ROS [15], etc. Hence, the bag file implementation of FLS flight paths enables different systems to inter-operate.

Third, we open source our software and the rose point cloud sequence as a bag file. See https://github.com/flyinglightspeck/FLSbagfile.

To the best of our knowledge, our conceptual and physical models of FLS flight paths are novel and have not been described elsewhere. The rest of this paper is organized as follows. Section 2 presents a conceptual design of FLS flight paths. While Section 3 introduces the bag file format, Section 4 presents an implementation of the conceptual model using this format. We evaluate this implementation in Section 5. Brief words of future research are offered in Section 6.

## 2 CONCEPTUAL MODEL AND LOGICAL OPERATIONS

We use the entity-relationship (ER) data model [1, 18] to represent animations, slide shows, point clouds and the FLS flight path data conceptually [1, 18]. This data model consists of entity sets (rectangles) and relationship sets (diamonds), see Figure 2. Both may have attributes denoted as ovals. A multi-valued attribute is represented as a double oval, e.g., Interval attribute of the Flight-Paths. Two or more entity sets may participate in a relationship set. This is denoted as a line from a rectangle to a diamond. A double line denotes total participation. This means all entities must participate in a relationship. One may use an aggregated entity set (dotted rectangle) with a relationship as its member. This aggregated entity set may participate in a relationship with other entity sets.

Figure 2 shows the ER diagram of FLS flight paths consisting of eight entity sets (rectangles) and four relationship sets (diamonds). It shows an Animation entity set consists of scenes. Each scene consists of point clouds. A Slide Show may also consist of point clouds. The many-to-many "Consists of" relationship between scenes (slide shows) and the point clouds identifies the sequence of point clouds. We aggregate each such a relationship as an entity that participates in the relationship set "Flight Paths". There is also a line from the

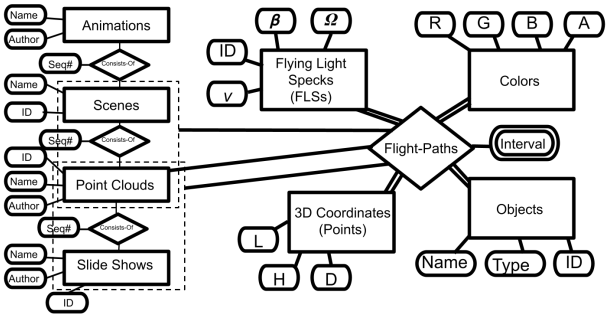


**Figure 2: A conceptual (ER) model of FLS flight paths.**

Point Clouds entity set to "Flight Paths", describing the illumination of a point cloud. Next, we detail the "Flight Paths" relationship set.

Flight-Paths is a many-to-many relationship set. It requires total participation of the following entity sets: Flying Light Specks (FLSs), 3D Coordinates, Colors, and Objects. The FLSs entity set describes the velocity model of an FLS ($v$), its flight time on a fully charged battery ($\beta$), and the time to charge its battery ($\Omega$). There may be a large number of FLSs with similar characteristics. An ID attribute is used to uniquely identify each FLS. The 3D Coordinates entity set identifies the Length $L$, Height $H$, and Depth $D$ coordinate[3] for an FLS. The Colors entity set identifies the intensity of different lights that the FLS must render. We are using the Red, Green, Blue, and Alpha (RGBA) color model in Figure 2. An FLS at its assigned 3D coordinate may be dark with no color. This is a special case with different interpretations by different applications. For example, an application may interpret dark as the color black, i.e., (0,0,0,0) with the RGBA color model. (The Objects entity set is a place holder for future work and described in Section 6.)

The *interval* attribute of Flight-Paths describes how long an FLS renders a color at a 3D coordinate. It may describe the duration either in time or point cloud ids. The latter identifies the start and end point cloud ids that identify a sub-sequence in a sequence. A system may translate from time to point cloud ids and vice versa given the number of point clouds rendered per second.

Interval is multi-valued (double ellipsoid) because an FLS may be required to render the same color at the same 3D coordinate at different times (or point cloud sub-sequences). An example is a rotating earth where the same positions and colors may be required for multiple rotations of the globe.

*Definition 2.1.* A closed interval [s, e] specifies the start and end time of an FLS rendering a color at a 3D coordinate. Values of s and e may identify the point cloud ids in the sequence that contains the point illuminated by the FLS. Alternatively, they may represent the start and end time relative to the start of a sequence. One may translate between the two using the number of point clouds rendered per unit of time.

*Definition 2.2.* An FLS flight path consists of at least three attributes: the 3D coordinates visited by the FLS, the color(s) rendered at a coordinate, and an interval for this rendering.

[1]Changes from one point cloud to the next point cloud of a scene.
[2]Changes from the last point cloud of Scene $i$ to the first point cloud of Scene $i + 1$.

[3]We do not use the X, Y, Z coordinate system because there is no consensus between mathematicians and the motion industry on depth. It is trivial to convert from $L$, $H$, $D$ to either definition.

The model and its bag file implementation are designed to support the following logical operations:

(1) Retrieve the flight path of FLSs to render a point cloud in a scene.
(2) Retrieve the flight path of FLSs to render a scene.
(3) Retrieve the flight path of FLSs to render a range of point clouds in a scene.
(4) Retrieve a point cloud given its id in a scene.

To illustrate the utility of these operations, consider an animated sequence consisting of S scenes. A wall mounted display may illuminate the first point cloud of each scene. A user may select one of these illuminations to render the corresponding scene. These are made possible by Operations 1 and 2, respectively. With Operation 1, the system fetches the FLS attributes such as flight path and color from the bag file for the 1st point cloud of each scene. Once the user selects a scene, the system fetches the FLS attributes for the entire scene to render the scene, Operation 2.

To render a scene, a point cloud, or a range of point clouds, the system fetches the FLS attributes such as flight path and color from the bag file. This information is used by the FLSs to illuminate the requested data.

Fetching a point cloud, Operation 4, enables a system designer to troubleshoot an illumination that does not look correct. By reconstructing the point cloud from the FLS flight paths, the designer may compare the point cloud with its original. If they do not match then the computed flight paths for the FLSs are erroneous and the software that generated the paths must be debugged. If they match then the original point cloud must be analyzed in the context of FLSs rendering them and re-drawn/adjusted as necessary.

## 3 OVERVIEW OF BAG FORMATTED FILES

A ROS bag file consists of a sequence of records. Each record has a header and data portion. The header identifies the record as either a bag `header`, `chunk`, `connection`, `message` data, `index` data, or `chunk-info`. This header is followed by the data for the record. We describe the different record types in turn using the verbatim font to highlight its name.

The bag `header` stores information about the entire bag, such as the offset to the first index data record, and the number of chunks and connections. A bag file has one bag header record and it is the first record in the file.

A `chunk` data consists of message data and connection records. These may be compressed using the method specified in the chunk record header.

A `connection` record header describes the topic on which the messages are stored. With ROS, topics are named communication lines for processes to implement publish/subscribe semantics. One or more processes subscribe to a topic to receive messages. A process publishes messages on the topic. The connection header also includes a unique connection ID.

A `message` data header specifies a connection ID and the time the message was received. Its data is the serialized message data.

An `index` data header specifies its connection ID and the number of index entries I. Its data consists of I repeated occurrences of timestamps, chunk record offsets, and message offsets. Each occurrence is an index entry.

Finally, `chunk-info` header specifies the offset of the chunk record, the start and end timestamps of the first and last message in the chunk, and the number of connections (C) in the chunk. Its data consists of C repeated occurrences of connection ID and the number of messages that arrived on this connection.

## 4 BAG FILE IMPLEMENTATION

This section describes a simple implementation of a subset of the ER model of Figure 2 using bag files and the algorithms that implement the logical operations of Section 2.

Our current implementation consists of records of `chunk` type for scenes and `message` records that store flight path of FLSs. A chunk may contain FLS flight paths (i.e., messages) for either one or multiple scenes. This is because the bag file format supports a chunk storing messages belonging to different topics.

Records of type `message` store the flight path of an FLS that renders a scene or a slide show, including its visited coordinates and rendered colors. Their header contains the identity of the scene, slide show, or point cloud rendered by the FLS. The FLS may or may not render a color at a coordinate. It is also possible for the FLS to stay at a 3D coordinate and render different colors. This information is stored in the record's body using the following simple data structures: An array of 3D coordinates, an array of colors, an array of intervals, and an array of bytes that indicates what information is present (WiP). The WiP array may contain the following three possible ascii values:

- 'B': The FLS flies to a new 3D coordinate and renders a new color. Both the 3D coordinate and color array elements are present.
- 'D': The FLS flies to a new 3D coordinate and renders the same color as in the previous location. Only the 3D coordinate array element is present.
- 'C': The FLS remains at the same 3D location and renders a new color. Only the color array element is present.

The WiP array de-duplicates the 3D coordinate and the color arrays, preventing them from repeating the same coordinate or color. The interval array identifies the duration of time an FLS is at a 3D coordinate and renders a certain color.

*Mapping to the ER model:* The `message` record corresponds to the Flight-Paths relationship set, and Colors and 3D Coordinates entity sets in Figure 2. The number of FLSs required to render a scene is the number of message records in its topic. An implementation of the rest of concepts shown in Figure 2 is future work. See Section 6.

### 4.1 Processing of Bag Files

Algorithm 1 implements the first 3 logical operations of Section 2. The fourth operation is implemented by Algorithm 2. We describe them in turn. Their complexity is presented in Section 5.

Algorithm 1 retrieves the FLS flight paths to render a sequence of point clouds in a scene. In addition to a bag file and a scene id, its inputs include start and end point cloud ids, spid and epid. Values of latter may fetch flight paths to render the entire scene, a point cloud in a scene, or a sequence of point clouds in a scene. Algorithm 1 fetches the FLS flight paths from the messages in the relevant chunks and generates an array of FLS flight paths for the specified range. Its output is an array of `message` records where

**Algorithm 1** sliceFLSFlightPath

**Require:** Bag file $f$, Scene id $sid$, starting and ending point cloud $spid$, $epid$
**Ensure:** One array of messages, each message is a FLS and its flight path

```
1: S ← Topic of bag file f corresponding to sid
2: n ← Number of messages in S    ▷ This is the number of FLSs
3: M ← []
4: for i ← 1 to n do
5:     Set WiP, Coords, Colors, Durs arrays to those in message (FLS) i
6:     Initialize newWiP, newCoords, newColors, newDurs to an empty array
7:     for j ← 1 to length of Durs do
8:         if NOT isClipStarted then
9:             maintain lastCoordinate and lastColor same as Algorithm 2 (11-18)
10:            if Durs[j].start ≤ id < Durs[j].end then
11:                isClipStarted ← true
12:                append 'B' to newWiP
13:                append lastCoordinate to newCoords
14:                append lastColor to newColors
15:                if Durs[j].start ≤ epid < Durs[j].end then
16:                    append Duration(spid, epid) to newDurs
17:                    break
18:                else
19:                    append Duration(spid, Durs[j].end) to newDurs
20:                end if
21:            end if
22:        else
23:            append WiP[j] to newWiP
24:            append Coords[j] to newCoords
25:            append Colors[j] to newColors
26:            if Durs[j].start ≤ epid < Durs[j].end then
27:                append Duration(Durs[j].start, epid) to newDurs
28:                break
29:            else
30:                append Duration(Durs[j].start, Durs[j].end) to newDurs
31:            end if
32:        end if
33:    end for
34:    m ← generate a ROS msg with newWiP, newCoords, newColors, newDurs
35:    append m to M
36: end for
37: return M
```

each record is a FLS flight path, i.e., WiP, 3D coordinates, and color arrays. One may adjust this design and its implementation to be an iterator to produce the qualifying `message` records one at a time [2].

.

Algorithm 2 processes a bag file to retrieve a single point cloud. Its inputs are a bag file, the identity of a scene, and the id of a point cloud in that scene. A scene is identified by its topic and assigned a unique connection ID. Either may be used with Algorithm 2.

**Algorithm 2** getPointCloud

**Require:** Bag file $f$, Scene id $sid$, and $id$ of a point cloud
**Ensure:** One point cloud

```
1: S ← Topic of bag file f corresponding to sid
2: n ← Number of messages in S    ▷ Number of FLSs
3: PtCld ← {}
4: for i ← 1 to n do
5:     WiP ← array of What-is-Present of scene S and FLS i
6:     Coords ← array of coordinates of scene S and FLS i
7:     Colors ← array of colors of scene S and FLS i
8:     Durs ← array of duration intervals of scene S and FLS i
9:     pt ← [ ]
10:    for j ← 1 to length of Durs do
11:        if WiP[j] is 'B' then
12:            lastCoordinate ← Coords[j]
13:            lastColor ← Colors[j]
14:        else if WiP[j] is 'C' then
15:            lastColor ← Colors[j]
16:        else
17:            lastCoordinate ← Coords[j]
18:        end if
19:        if Durs[j].start ≤ id < Durs[j].end then
20:            append lastCoordinate to pt
21:            append lastColor to pt
22:            break
23:        end if
24:    end for
25:    add pt to PtCld
26: end for
27: return PtCld
```

## 5 AN EVALUATION

The number of disk (SSD) block reads dominates the execution time of Algorithms 1 and 2. The number of blocks to read messages that constitute a scene is dictated by the number of fetched messages and their size. Lossless compression technique enhance performance by maximizing the amount of data stored in a block.

The bag file implementation may use a lossless compression technique in a variety of ways: lz4 with chunks, Gzip for the entire file, or both together. This section quantifies their tradeoffs. An interesting find is that bag+lz+gzip does not result in the smallest file size. See discussions of Tables 1 and 2. As a comparison yardstick, we use the archival tool TAR with and without Gzip. We evaluate the following alternatives:

(1) TAR: An archive of the directory of point cloud files from the rose point cloud sequence. The directory consists of a fixed number of point cloud files.
(2) TAR+Gzip: Compress tar archive file of the point cloud directory of Experiment 1 using Lempel-Ziv coding LZ77.
(3) Bag: Implementation of the model described in this paper, see Sections 2 and 3.
(4) Bag+lz4: Compress chunks of the bag file from Experiment 3 using lz4.
(5) Bag+Gzip: Compress the bag file of Experiment 3 using Lempel-Zip coding LZ77.

**Table 1: Size of alternative physical representations in Megabytes.**

| # of Point Clouds | TAR | TAR+Gzip | Bag | Bag+lz4 | Bag+Gzip | Bag+lz4+Gzip |
|---|---|---|---|---|---|---|
| 10 | 28.1 | 10.3 | 8.0 | 2.3 | 1.3 | 1.4 |
| 20 | 56.1 | 20.6 | 8.5 | 2.6 | 1.6 | 1.7 |
| 40 | 112.1 | 41.1 | 9.5 | 3.2 | 2.1 | 2.2 |
| 80 | 224.3 | 82.2 | 11.5 | 4.4 | 3.2 | 3.2 |
| 100 | 280.3 | 102.8 | 12.5 | 5.1 | 3.7 | 3.8 |
| 110 | 308.3 | 113.1 | 13.0 | 5.4 | 4.0 | 4.1 |
| 115 | 322.3 | 118.2 | 13.1 | 5.5 | 4.0 | 4.1 |

**Table 2: Compression ratios.**

| # of Point Clouds | $\frac{TAR}{Bag}$ | $\frac{TAR}{Bag+lz4}$ | $\frac{TAR+Gzip}{Bag}$ | $\frac{TAR+Gzip}{Bag+lz4}$ | $\frac{TAR+Gzip}{Bag+Gzip}$ | $\frac{TAR+Gzip}{Bag+lz4+Gzip}$ |
|---|---|---|---|---|---|---|
| 10 | 3.5 | 12.2 | 1.3 | 4.5 | 7.9 | 7.36 |
| 20 | 6.6 | 21.6 | 2.4 | 7.9 | 12.9 | 12.12 |
| 40 | 11.8 | 35.0 | 4.3 | 12.8 | 19.6 | 18.68 |
| 80 | 19.5 | 51.0 | 7.1 | 18.7 | 25.7 | 25.69 |
| 100 | 22.4 | 55.0 | 8.2 | 20.2 | 27.8 | 27.05 |
| 110 | 23.7 | 57.1 | 8.7 | 20.9 | 28.3 | 27.59 |
| 115 | 24.6 | 58.6 | 9.0 | 21.5 | 29.6 | 28.83 |

(6) Bag+lz4+Gzip: Compress the bag file of Experiment 4 using Lempel-Zip coding LZ77.

Each experiment produces a single file as a function of the number of point clouds from the rose point cloud sequence. The size of this file is reported in Table 1. The columns of this table name the experiments. The rows of this table identify the number of point clouds from the rose point cloud sequence. This sequence consists of 115 point clouds which is the last row of Table 1.

Obtained results highlight the following lessons. Bag files are more compact than the original point cloud sequence. They represent flight path of FLSs. When an FLS stays in the same location for multiple point clouds then it is represented with one coordinate and an interval. The point cloud sequence repeats this coordinate, resulting in a larger size.

With the rose point cloud sequence, the size of the bag file increases insignificantly (< 2x) as we increase the number of point clouds more than 10x, from 10 to 115. Each point cloud consists of 65K points. 2009 points change from one point cloud to the next. These changes dictate the increase in flight path of FLSs and the size of the bag file as a function of the number of point clouds.

When comparing lz4 with Gzip, Gzip produces smaller bag files, compare Bag+lz4 and Bag+Gzip columns of Table 1. We evaluated alternative explanations for this observation such as lz4 processing a chunk versus Gzip processing an entire bag file, and the difference in the algorithm used by Gzip (Lempel-Ziv coding LZ77) versus lz4. To eliminate the first possibility, we repeated an experiment by generating a bag with a 15 MB chunk size (instead of 768 KB chunk size). A 15 MB chunk size enables a bag file to store the messages of the 115 point cloud sequence in one chunk and apply lz4 to all messages. The resulting bag file is slightly smaller when compared with a 768 KB chunk size, 5,450,860 bytes with 15 MB compared with 5,458,654 bytes with 768 KB. Both are significantly larger than 4,030,024 bytes realized with Gzip. Hence, we conclude the algorithm difference as the explanation for the smaller file size.

Use of both lz4 and Gzip is inferior to Gzip by itself, producing a larger file size. Compare columns Bag+Gzip with Bag+Lz4+Gzip of Table 1. However, using Gzip with lz4 results in a smaller file. Compare columns Bag+lz4 and Bag+lz4+Gzip of Table 1.

One may consider an algorithm that generates the flight path of FLSs as an application specific lossless compression technique for point clouds. It produces a file that can be processed to retrieve the points in a point cloud sequence, see Algorithm 2. Table 2 shows the compression factors observed with the bag file when compared with the original point cloud sequence. The bag file is significantly smaller. 24.6x with the entire 115 point cloud sequence.

One may compress the archive containing the directory of the point cloud sequence, TAR+Gzip. This representation is more compact than the original archival file, TAR. The bag file is still more compact. 9x with the entire 115 point clouds. One may also compress the bag file using Gzip, realizing 29.6x reduction in size when compared with TAR+Gzip for the entire 115 point cloud sequence.

While the use of lz4 with a bag file is not as space efficient as Gzip, lz4 is applied as a part of generating the bag file. Gzip is an extra step that a user must perform to compact the bag file. Comparing Bag+lz4 with TAR shows a 58.6 reduction in size for the entire 115 point cloud sequence.

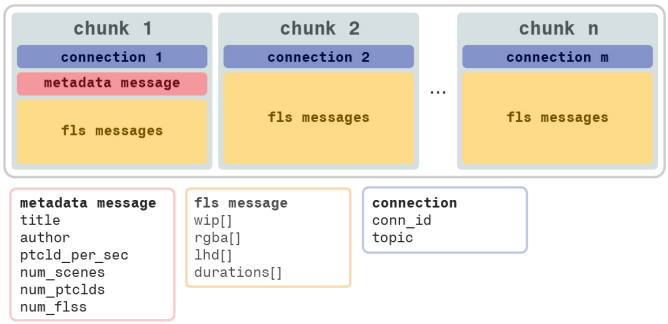


**Figure 3: Future organization of bag file for Figure 2.**

Use of lz4 with a bag file has the following advantage when compared with Gzip. An algorithm may seek into the lz4 compressed bag file, read a fraction of its data (say a message), decompress and process the fetched data. With Gzip, the same algorithm must first read the entire file and decompress it prior to seeking into it to fetch the relevant data, reading more blocks than lz4.

## 6 CONCLUSIONS AND FUTURE RESEARCH

The bag file representation of FLS flight paths has enabled us to inter-operate between software systems using MATLAB and Python programming languages. Several student groups have used them with AirSim [17] to investigate alternative FLS lighting designs.

We are extending our current implementation to support a larger number of concepts shown in Figure 2. A future bag file organization is shown in Figure 3. Its metadata `message` record contains the properties of an illumination. In addition, it includes the aggregation of the Consists-Of relationship set between Scenes and the Point Clouds entity sets, e.g., num_scenes, num_ptclds, etc. Similar to our current implementation, its `chunk` records contain messages that describe the flight path of FLSs to render an illumination. The $n$ `connection` records identify the $n$ different scene topics and their unique connection IDs, conn_ids.

More longer term, we plan to extend the implementation to represent objects and characters that constitute a scene. The conceptual data model of Figure 2 identifies an object entity set and the FLS flight paths that render it in a scene. One approach to represent this information is to extend `message` records describing FLS flight paths to identify an object. It may maintain additional metadata to identify the different objects and the starting offset of the chunk that contains them in a scene.

In addition to bag files, we will explore alternative file formats to represent the conceptual model of Figure 2. One possibility is a simple key-value representation. It includes a key with the metadata of a file as its value. This metadata identifies the different keys and types of values. For example, a key may identify a scene, an FLS flight path, an aggregation such as the number of scenes, etc. This may be more intuitive than using bag file constructs. For example, use of conn_id as the scene id may not be intuitive. A challenge with a new file format is the requirement for software to read and write its data with different programming languages, e.g., Python, and frameworks, e.g., MATLAB. We will explore this possibility as we implement a cuboid 3D display that renders a scene using FLSs.

## ACKNOWLEDGMENTS

This research was supported in part by the National Science Foundation grant IIS-2232382.

## REFERENCES

[1] Peter P. Chen. 1975. The Entity-Relationship Model: Toward a Unified View of Data. In *Proceedings of the International Conference on Very Large Data Bases, September 22-24, 1975, Framingham, Massachusetts, USA*, Douglas S. Kerr (Ed.). ACM, 173. https://doi.org/10.1145/1282480.1282492

[2] D. DeWitt, S. Ghandeharizadeh, D. Schneider, A. Bricker, H. Hsiao, and R. Rasmussen. 1990. The Gamma Database Machine Project. *IEEE Transactions on Knowledge and Data Engineering* 1, 2 (March 1990).

[3] Tech Force. 2019. 500 Drones Create Stunning Light Show on AI-Driven Future, May 19, 2019. See https://www.youtube.com/watch?v=LvYNHSf7FbI.

[4] Shahram Ghandeharizadeh. 2021. Holodeck: Immersive 3D Displays Using Swarms of Flying Light Specks. In *ACM Multimedia Asia* (Gold Coast, Australia). 1–7. https://doi.org/10.1145/3469877.3493698

[5] Shahram Ghandeharizadeh. 2022. Display of 3D Illuminations using Flying Light Specks. In *ACM Multimedia* (Lisboa, Portugal) *(MM '22)*. ACM, New York, NY, USA. https://doi.org/10.1145/3503161.3548250

[6] Shahram Ghandeharizadeh and Luis Garcia. 2022. Safety in the Emerging Holodeck Applications. In *CHI 2022 Workshop on Novel Challenges of Safety, Security and Privacy in Extended Reality*. ACM, New York, NY, USA.

[7] Danillo Bracco Graziosi, Ohji Nakagami, Shinroku Kuma, Alexandre Zaghetto, T. Suzuki, and Ali J. Tabatabai. 2020. An Overview of Ongoing Point Cloud Compression Standardization Activities: Video-based (V-PCC) and Geometry-based (G-PCC). *APSIPA Trans. on Signal and Information Processing* 9 (2020).

[8] Ravinder Kumar Jyoti, Mohit Kumar Malhotra, and Debasish Ghose. 2021. Rogue Agent Identification and Collision Avoidance in Formation Flights using Potential Fields. In *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*. 1080–1088. https://doi.org/10.1109/ICUAS51884.2021.9476866

[9] O. Khatib. 1985. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. In *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, Vol. 2. 500–505. https://doi.org/10.1109/ROBOT.1985.1087247

[10] N. Koenig and A. Howard. 2004. Design and use Paradigms for Gazebo, an Open-Source Multi-Robot Simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, Vol. 3. 2149–2154 vol.3. https://doi.org/10.1109/IROS.2004.1389727

[11] MathWorks. https://www.mathworks.com/help/ros/ref/rosbagwriter.html. Create and Write Logs to ROSbag Log File, rosbagwritter.

[12] Thulasi Mylvaganam, Mario Sassano, and Alessandro Astolfi. 2017. A Differential Game Approach to Multi-Agent Collision Avoidance. *IEEE Trans. Automat. Control* PP (04 2017), 4229–4235. https://doi.org/10.1109/TAC.2017.2680602

[13] ROS.org ROS 2 Documentation. http://wiki.ros.org/Bags/Format/2.0. Version 2.0 of the ROS Bag File Format.

[14] ROS.org. https://www.ros.org/. Robot Operating System.

[15] ROS.org. http://wiki.ros.org/rosbag/Code API#cpp_api. ROS Code API.

[16] Sebastian Schwarz, Marius Preda, Vittorio Baroncini, Madhukar Budagavi, Pablo César, Philip A. Chou, Robert A. Cohen, Maja Krivokuća, Sebastien Lasserre, Zhu Li, Joan Llach, Khaled Mammou, Rufael Mekuria, Ohji Nakagami, Ernestasia Siahaan, Ali J. Tabatabai, Alexis M. Tourapis, and Vladyslav Zakharchenko. 2019. Emerging MPEG Standards for Point Cloud Compression. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9 (2019), 133–148.

[17] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. 2018. AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. In *Field and Service Robotics*, Marco Hutter and Roland Siegwart (Eds.). Springer International Publishing, Cham, 621–635.

[18] Abraham Silberschatz, Henry Korth, and S. Sudarshan. 2020. *Database System Concepts. Chapter 6.* (7 ed.). McGraw Hill, USA.

[19] Hang Sun, Juntong Qi, Chong Wu, and Mingming Wang. 2020. Path Planning for Dense Drone Formation Based on Modified Artificial Potential Fields. *39th Chinese Control Conference (CCC)* (2020), 4658–4664.

[20] Jiayi Sun, Jun Tang, and Songyang Lao. 2017. Collision Avoidance for Cooperative UAVs With Optimized Artificial Potential Field Algorithm. *IEEE Access* PP (08 2017), 18382–18390. https://doi.org/10.1109/ACCESS.2017.2746752

[21] Ternaris. https://pypi.org/project/rosbags/. Pure Python Library to Read, Modify, Convert, and Write ROSbag Files, rosbags 0.9.12.