The ASTM E57 File Format for 3D Imaging Data Exchange

Daniel Huber
The Robotics Institute, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213

ABSTRACT

There is currently no general-purpose, open standard for storing data produced by three dimensional (3D) imaging systems, such as laser scanners. As a result, producers and consumers of such data rely on proprietary or ad-hoc formats to store and exchange data. There is a critical need in the 3D imaging industry for open standards that promote data interoperability among 3D imaging hardware and software systems. For the past three years, a group of volunteers has been working within the ASTM E57 Committee on 3D Imaging Systems to develop an open standard for 3D imaging system data exchange to meet this need. The E57 File Format for 3D Imaging Data Exchange (E57 format hereafter) is capable of storing point cloud data from laser scanners and other 3D imaging systems, as well as associated 2D imagery and core meta-data. This paper describes the motivation, requirements, design, and implementation of the E57 format, and highlights the technical concepts developed for the standard. We also compare the format with other proprietary or special purpose 3D imaging formats, such as the LAS format, and we discuss the open source library implementation designed to read, write, and validate E57 files.

Keywords: 3D image, laser scan, LIDAR, LADAR, point cloud, file format, standard, open source

1. INTRODUCTION

Three dimensional (3D) imaging systems, such as laser scanners, produce large data sets of 3D point measurements, which are often referred to as point clouds (Figure 1). There is currently no general-purpose, open standard for storing such data. As a result, producers and consumers of 3D imaging system data rely on proprietary or ad-hoc formats to store and exchange data. Information stored in proprietary formats can be difficult to access, and ad-hoc formats increase

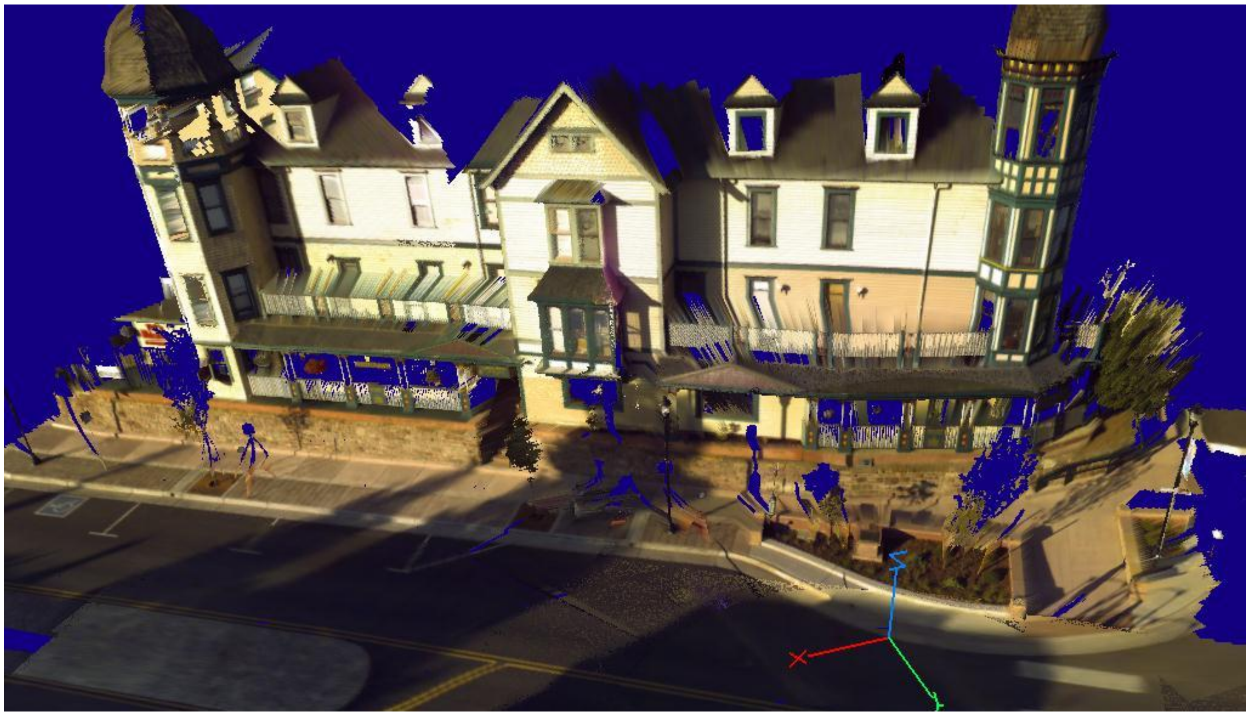

Figure 1. An example data set that can be encoded in an E57 file. The data contains several laser scans obtained from different locations. (Image courtesy of Intelisum, Inc.)

software development costs and are not easily extended to widespread usage. There is a critical need in the 3D imaging industry for open standards that promote data interoperability among 3D imaging hardware and software systems.

For the past three years, a group of volunteers has been working within the ASTM E57 Committee on 3D Imaging Systems to develop an open standard for 3D imaging system data exchange [1]. The E57.04 Sub-committee on Data Interoperability is part of the ASTM International, a global standards organization. This committee includes representatives from many of the major 3D imaging system manufacturers, 3D imaging software vendors, and 3D imaging service providers, as well as industry consultants and academic researchers. The result of the efforts of this committee has been the E57 File Format for 3D Imaging Data Exchange (E57 format hereafter), which was recently approved as ASTM standard E2761.

The E57 format is designed to be a general purpose, open standard for storing data produced by 3D imaging systems. The file format is capable of storing point cloud data from laser scanners, but can also encode data from flash LIDAR systems, structured light 3D scanners, stereo vision systems, and other devices that produce 3D measurements. In addition to storing 3D point measurements, the format can store associated 2D imagery, such as that produced by a digital camera, as well as core meta-data associated with the 2D images and 3D points. As a proof of concept of the E57 file format, an open source reference library has been developed that is capable of reading, writing, and validating files in the E57 format.

This paper will describe the development of the E57 file format from its inception through its acceptance as a standard. We first discuss the guiding principles behind the design and the formulation of those principles into design requirements (Section 2). Next, we outline the main features of the design, including the ability to encode extremely large files – up to 9 exabytes in length, the ability to encode organized "gridded" point cloud data as well as unorganized data, and the ability to extend the format to support special-purpose needs, such as aerial sensing (Section 3). Next, we describe the design and implementation of the reference library (Section 4). We then discuss how the E57 format compares to other 3D imaging file formats, including the LAS format, which is a special-purpose format developed by the American Society for Photogrammetry & Remote Sensing (ASPRS) for aerial sensing applications (Section 5). We conclude with a summary of open issues that will be addressed in the future versions of the standard, including uncertainty representation and mobile sensing (Section 6).

2. 3D IMAGE FILE FORMAT REQUIREMENTS

The requirements for the E57 file format were developed by consensus of the participating representatives from industry (laser scanner hardware and software developers and consultants) and government. This section highlights the results of the requirements specification developed by this group. The committee determined that the format should be developed in a phased implementation, with an initial specification of core features that would enable widespread industry adoption of the standard followed by new versions of the specification that would incorporate the more challenging capabilities, such as advanced compression algorithms. The initial standard would focus on storing 3D data to support reliable data exchange between software applications, while, at the same time, making design decisions that would support requirements unique to archival storage of 3D data for incorporation in a future version of the standard.

The requirements for the standard are organized around five guiding principles:

1. *Reliable interoperability* – The data should be transferable from any vendor to any other vendor.
2. *Open* – The standard and its implementation should be made openly available and should be well documented, unrestricted, and vendor neutral.
3. *Low barrier for adoption* – The development cost for adopters should be kept to a minimum.
4. *Minimalist design* – The design should be kept as simple as possible, while achieving the required goals.
5. *Extensibility* – The design should be extendable so that new capabilities can be added in the future without breaking the core functionality.

These principals dictate that the type of information to be stored in a 3D image file should be kept to a minimum. The committee agreed that a file should store 3D point data in a flexible manner, which means that the format should be capable of handling unorganized point clouds, regularly organized "gridded" data sets, and data with multiple returns. The format should allow multiple data sets to be stored within a single file, but all of the data must be representable in a single coordinate system, for example, by storing associated pose information for each data set. Many sensors acquire imagery in conjunction with the 3D data, so the format should be capable of storing associated imagery as well as pose information needed to register the images with the 3D data.

In order to adhere to the minimalist principle, additional information that, while potentially useful to some downstream users, was determined to be outside the scope of the format. One example of such information is vendor-specific meta-data, such as calibration parameters. Vendor-specific corrections should be applied prior to storing in the file. The format is not intended to be a working data format or a project file format. As such, derived data, such as modeled objects created from the 3D data, should not be included. The extensibility of the standard means that information that is later deemed to be important can be represented by extending the core functionality of the standard. The standard should not seek to replace the native file formats of individual vendors, but instead is intended to represent the aspects of 3D data that are most useful in a simple, yet efficient, format.

The requirements also include a number of secondary goals:

- *Support for internationalization* – Strings should be representable in any language.
- *Support for extremely large file sizes* – Up to 2^{61} bytes in length.
- *Self-describing* – The standard should be self-documenting. For example, it should not require external lookup tables to interpret the meaning of specific fields.
- *Computer readable* – It should be possible to automatically verify that a file meets the syntactic requirements of the standard.
- *Speed and storage efficient* – The format file size should be significantly smaller than the equivalent file stored in ASCII format, and reading or writing a file should be significantly faster than reading or writing the equivalent file in ASCII format.
- *Memory efficient* – The memory requirements should allow for implementation of the standard on a microcontroller.
- *LAS compatibility* – The format should support the features of the ASPRS LAS format through an extension to the core functionality.

Once the requirements were in place and agreed upon by the committee members, the next step was to transform these requirements into a concrete design.

3. E57 FILE FORMAT DESIGN

Data file formats fall into two broad categories of design. One type of format employs fixed sized fields and records in a rigid format. The fields are frequently populated with special codes that can be interpreted only through the use of external lookup tables. The JPEG standard is an example of this type of approach. The second type of format uses flexible, self-documenting structures with variable lengths and structure. The VRML standard is an example of this type of approach. The former approach has the advantage of being fast to interpret and simple to understand, but has the

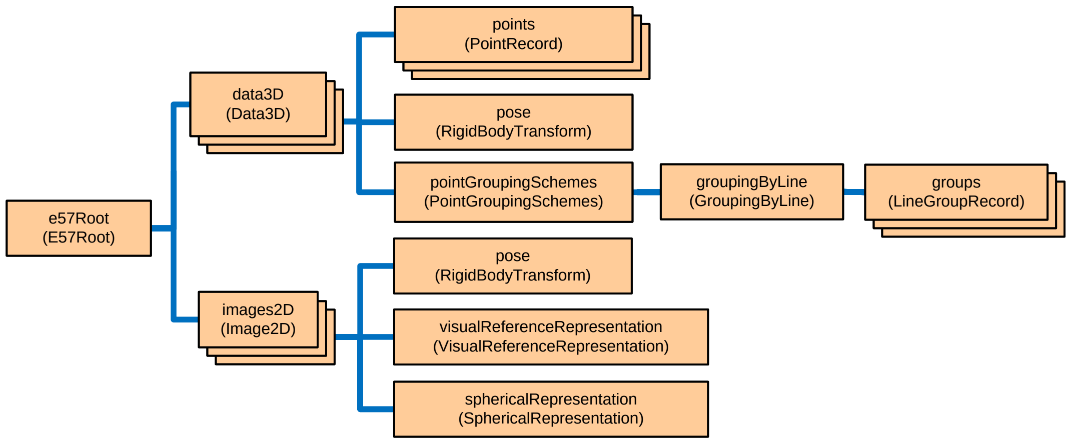

Figure 2. The data within an E57 file is stored in a hierarchical tree structure that is based on XML. An illustrative example structure is shown here.

disadvantage of not being very flexible. The latter approach is more flexible, but can lead to higher complexity and slower implementations. The self-documentation and extensibility requirements for the ASTM 3D format suggest the latter approach, while the simplicity, speed, and size requirements suggest the former approach. These competing requirements resulted in a design that is a hybrid of the two approaches, and the best aspects of each approach are included in the representation.

3.1 E57 File Structure

At a high level, the structure of an E57 file is a hierarchical tree structure (Figure 2). The format of the hierarchy is based on the XML data format. However, it would be inefficient to represent the extremely large data sets associated with laser scanners in a pure XML format. The file sizes would be unacceptably large, and data I/O would be painfully slow. Therefore, at a low level, the actual point data is represented using a compressed binary format. Other large data blocks, such as images, are also represented efficiently in binary. In this way, the format supports flexibility and extensibility using text-based XML, while enabling efficient I/O and storage using compressed streams of binary data.

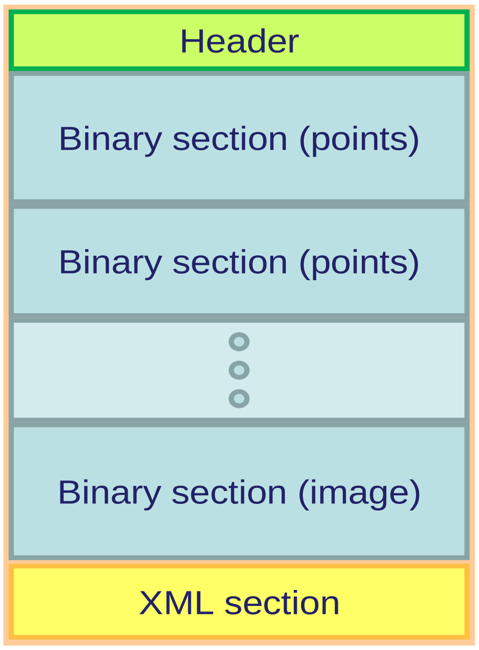

Figure 3. An E57 file consists of a short header, followed by zero or more binary sections for encoding the point data or images, and, finally, an XML section that stores the hierarchical data structure shown in Figure 2.

An E57 file is divided into three parts: a header, a set of optional binary sections, and an XML section (Figure 3). The header is a small, 48-byte binary structure that contains critical file-level information, such as the version number and the location of the XML section. The XML section contains the hierarchical tree structure describe above. If the file contains point data or images, these portions of the hierarchy are referenced by the XML section, and the actual data is stored in the binary sections, with a separate section for each set of points or image.

3.2 Error Checking

An error-checking mechanism is built directly into the E57 format. An E57 file is divided into a set of pages, each 1024 bytes in size. The first 1020 bytes contains data, and the last 4 bytes contain a cyclic redundancy check (CRC) checksum. When encoding an E57 file, the data to be encoded is divided into 1020 byte chunks, a CRC checksum is computed for that data, and the concatenated data and checksum are stored in the file. When decoding an E57 file, the reverse process occurs, and the checksum for each page of data is verified against the stored checksum. In this way, the format can detect if portions of the file have been corrupted during storage or transmission. While this error checking adds some complexity to the encoding and decoding of the format, the benefit is that if portions of the file are corrupted, it may still be possible to extract useful data from uncorrupted parts of the file. While it is true that many E57 users will have hardware level or file system level mechanisms to prevent data corruption, this mechanism serves as a second line of defense against data loss.

3.3 The XML Hierarchy

The XML section of an E57 file describes the data hierarchy using a subset of standard XML. The hierarchy is built up using a set of core building blocks, known as E57 elements. The standard defines eight E57 element types, five of which are terminal types, and three of which are non-terminal. The terminal types correspond roughly to data types that are commonly found in database or programming languages. Terminal types include Integer, Float, ScaledInteger, String, and Blob.

- *Integer* – Stores a signed integer up to 63 bits in size. For this, and all numerical types, the user can specify the range of possible values that can be stored.
- *Float* – Stores a single or double precision floating point number in IEEE 754-1985 format. Some special cases, such as NaN, are disallowed.

- *ScaledInteger* – Stores a number with a fractional part as an integer along with a scale and offset. The actual data value is computed by multiplying the stored integer by the scale and adding the offset. This concept is similar to a fixed point representation, and it allows users to balance storage requirements with data resolution requirements. For example, if range data is only accurate to one centimeter over a range of ten meters, the values could be stored in 10-bit scaled integers with the scale factor of 0.01. Not only is the number of bits substantially less than a single precision floating point number, but it is potentially easier to compress integer values than floating point values.

- *String* – Stores text data. The text is encoded using UTF-8 to ensure support for international characters.

- *Blob* – Stores an opaque block of binary data that is interpreted by the reader depending on the context. For example, in the standard, blobs are used to store image data. Associated with each blob in the XML section of an E57 file is a corresponding blob binary section in the file. The XML representation of a blob contains a pointer to the position in the file where the blob binary data is stored.

Non-terminal types act as containers, and may contain children that are other terminal or non-terminal E57 elements. Non-terminal types include Structure, Vector, and CompressedVector.

- *Structure* – Contains an unordered set of E57 elements of any type. An E57 structure is analogous to a structure in the C programming language.

- *Vector* – Contains an ordered list of substantially identically typed items. A flag in the element allows the items to be restricted to be exactly identically typed if desired. In some cases, however, it is beneficial to allow some flexibility in the types of the Vector's children or more remote descendents. For example, if the items in a Vector are Structures, and if the Structure is defined to have optional child elements, then a user would not be required to include a particular optional child element in every item just because it is included in one item.

- *CompressedVector* – Contains an ordered list of identically typed items, which are compressed in binary format. As with the Blob data type, the representation of a CompressedVector is divided into an XML portion and a binary portion. The XML part of a CompressedVector contains a pointer to the position in the file where the corresponding binary section is located. The XML representation also contains a prototype structure that describes the fields that are stored in the binary data. Finally, the XML representation specifies the manner in which each leaf node in the prototype is actually compressed.

The elements in the XML section of an E57 file must follow a particular format. Different data types are defined by the standard to support the storage of 3D points and 2D images in a common, file-level coordinate system. The root node of the data hierarchy is a structure called the E57Root. This structure contains a vector for storing an arbitrary number of 3D data sets (e.g., point clouds) and another vector for storing 2D images (e.g., camera images). While it is possible to store multiple 3D data sets and images in a single file, all of the data is required to be represented in a single, common coordinate system. This requirement can be met either by placing all data explicitly in a file-level coordinate system or through the use of rigid body transforms that indicate how to transform individual data sets into the file-level coordinate system. The E57Root structure also contains file-level information that is not already stored in the header, such as the creation date and time. Finally, the structure contains an optional coordinate metadata string for encoding a geodetic datum and related information necessary for referencing a data set in a standardized Coordinate Reference System (CRS). The standard uses the specification developed by the Open Geospatial Consortium for this purpose [2].

3.4 Point Data

Each set of 3D points is stored in a separate structure known as a Data3D. This structure contains the points themselves, in the form of a CompressedVector. The fields associated with each point are defined in a flexible manner. An E57 file writer can define whatever fields are produced by a given sensor and can encode them in a manner that is most suitable to the sensor type, or even the individual data set. For example, if a sensor produces points in spherical coordinates and also intensity values, then only those fields need to be included. The range and resolution of each field can be tailored to the needs of the individual application. A sensor that produces color measurements that are 10 bits can use just 10 bits to represent the measurements. The standard defines a broad set of fields that are in common use, including Cartesian and spherical point coordinates, row and column indices, return number, intensity, color, and time stamps. These pre-defined fields enable the format to represent unordered 3D data – often known as point clouds – as well as data that is organized into gridded rows and columns. The structure also handles multiple return and no return situations. If the pre-defined fields for point data are not sufficient for some application, it is straightforward to add new fields using the standard's extension mechanism (Section 3.7).

Point data can also be divided into logical groupings. The standard defines one type of grouping – grouping by line. This type of grouping is useful for data that consists of a sequence of rows or columns, which is typical of many commercial laser scanners. Line groups enable a file reader to directly seek to and read a specific line in the data file without having to read all the lines of data into memory first. Other types of point groups could be used, for example, to define class label groupings over the data, such as "grass," "road," and "water." Although such a classification mechanism is not defined in the standard, the extension mechanism can be used to define this type of grouping.

The E57 format includes a mechanism to allow individual 3D data sets to be stored in the local coordinate system of the scanner while still fulfilling the requirement that all the data be representable in a file-level coordinate system. Each 3D data set includes an optional pose element that stores a rigid body transform, which, when applied to the data points, will place them in the file-level coordinate system.

3.5 Images

Two dimensional images are the second significant type of information that may be stored in an E57 file. Frequently, 3D imaging sensors will capture imagery, either using built in color sensors or through a conventional digital camera. These images can be a valuable asset for visualization or analysis of 3D data sets. Images are most useful if they are calibrated with respect to the 3D point data so that individual 3D points can be projected onto the images (or, conversely, the images can be projected onto the 3D point data). To support this functionality, the standard defines four different image representations. These representations correspond to different common camera models.

- *Visual reference* – This representation is used for uncalibrated images. In this case, the images can be viewed, but there is no expectation of being able to map between 3D points and image pixels.
- *Pinhole* – This representation stores a pinhole camera projection model (Figure 4). Digital cameras with typical, non-fisheye, lenses are well-approximated by this model.
- *Spherical* – This representation stores a spherical camera projection model. Images from fisheye lenses or image mosaics generated from a single position can be represented using this model.
- *Cylindrical* – This representation stores a cylindrical camera projection model. Images from a rotating single line scanning camera can be represented using this model.

In order to keep the models used in the representations as simple as possible, the standard requires that certain types of distortion (such as radial and tangential distortion in the pinhole representation) be removed prior to storing in an E57 file. This decision is sensible because there is no universally accepted model for distortion, but the downside is that the undistortion process can produce non-rectangular images. The standard addresses this by introducing a binary mask to indicate invalid pixels in a rectangular image that encompasses the non-rectangular undistorted image. The images themselves are stored as Blob elements, either in JPEG or PNG format.

3.6 Binary Encoding

The bulk of an E57 file will typically be encoded in the binary sections. There are two types of binary sections – Blob sections and CompressedVector sections.

Blob sections contain the binary portion of Blob E57 elements. This data is interpreted based on the context in which the Blob element is defined. For example, if the context of the Blob indicates that the data should be a JPEG image, then a reader would interpret the binary data as if it were a separate JPEG image file. In this way, the file format can encapsulate other

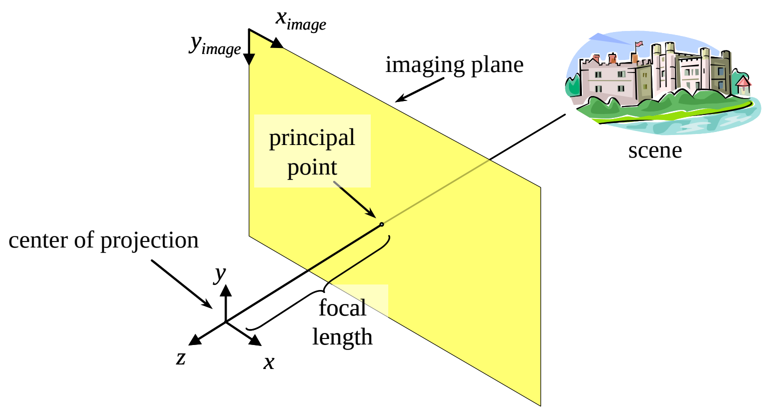

Figure 4. The pinhole camera model is one of four image representations supported by the E57 format. The model allows 3D points to be projected onto camera images and image pixels to be projected into 3D space.

existing standardized file formats without requiring detailed knowledge about the specifics of that format. In the case of a Blob containing a JPEG image, all that would be needed is a library capable of reading a JPEG file as a data stream, and an E57 file reader could stream the relevant data to the JPEG reader and then return the resulting image.

CompressedVector sections contain the binary portion of CompressedVector E57 elements. A CompressedVector section consists of a sequence of variable length binary sequences called packets. There are three types of packets – index packets, data packets, and ignored packets. Ignored packets are used for reserving space for later use. Since binary sections occur before the XML section in an E57 file, this feature can be helpful when data is obtained incrementally.

Efficient access to items stored in a CompressedVector is achieved through an indexing mechanism. A CompressedVector stores a sequentially numbered set of identically typed items, known as records. These records are organized into sequential groups, known as chunks. Index packets form a database of the beginning locations of chunks in a CompressedVector. This database allows rapid access to a specific record number, which, while not random access, is generally significantly more efficient than sequentially reading through the file to reach a desired record. Index packets are organized into a tree that is similar in spirit to the inode data structure used in many Unix file systems. The leaf nodes in the index packet tree contain the locations of the beginning of each chunk.

The data in a CompressedVector is stored in data packets. Different coder-decoders (codecs) can be defined to convert the fields from a sequence of records into a serial data stream and back. The standard includes a simple codec called the bitpack codec. The next version of the standard is intended to include codecs with more sophisticated compression mechanisms. The bitpack codec is a lossless compression algorithm that is primarily useful for reducing the size of Integers and ScaledIntegers. The codec uses the defined range (minimum and maximum values) of a given Integer or ScaledInteger field to determine the number of bits that will be required to store any value within that field. For example, an Integer field with values between 0 and 7 would only require 3 bits per item. The XML portion of the CompressedVector contains a mapping between fields and codecs, which allows multiple related fields to be grouped together for compression.

3.7 Extensions

The E57 format design includes a mechanism for extending the format with new capabilities while maintaining both forward and backward compatibility. Extensions can be used to define special-purpose elements or experimental capabilities. If an extension is well-accepted by the user community, a future revision of the standard would likely incorporate the extension into its core feature set. An extension might define, for example, the additional elements needed to encode LAS files in the E57 format. Another extension could define elements for representing mesh connectivity for the points in a point cloud.

It is important that new features should not prevent older software from reading and using files encoded using extended features. Extensions are defined using XML namespaces, which uniquely identify any new element names defined by the extension. When an E57 file reader encounters an element with a namespace that it does not understand, it simply ignores the unrecognized element and any of its child elements. In this way, older readers can continue to read and process E57 files that include new extensions, though without the benefit of the new capabilities defined by the new extensions.

4. E57 FILE FORMAT IMPLEMENTATION

One of the goals of the standard was to minimize the cost and effort required to adopt the standard. To accomplish this goal, we have created a cross-platform, open source implementation of the standard that is intended to lower the barriers to adoption of the standard and to provide a reference to compare other implementations against [3]. The software, which is known as libE57, consists of a library, supporting utilities and example programs, and documentation. The software includes two separate application programming interfaces (APIs) for reading, writing, and manipulating E57 files – the Foundation API and the Simple API. The Foundation API is a full-featured interface that operates at a relatively low-level, allowing control over all aspects of an E57 file, including custom extensions. The Simple API is a simplified interface (built on top of the Foundation API) that supports the most common use cases for reading and writing E57 files. The library also includes tools for converting LAS format files into E57 files, and a tool for validating E57 file correctness is under development. The E57 software is currently undergoing beta testing.

Most of the major laser scanner manufacturers have pledged to integrate the E57 format into upcoming versions of their software tools. This list includes InteliSum, Inc., Leica Geosystems, Optech, Inc., Quantapoint, Inc., Riegl Laser

Measurement Systems GmbH, Trimble Navigation Ltd., and Zoller+Frö hlich GmbH. Since the committee designing the standard includes representatives from many of these companies, we anticipate that adoption among hardware vendors will be relatively rapid. Software vendors, government organizations, service providers, consultants, and academic institutions have also pledged support of the standard. A complete list of supporting partners is maintained on the libE57 web site [3].

5. DISCUSSION AND COMPARISON WITH OTHER 3D FILE FORMATS

While most users of 3D imaging systems recognize that there is a critical need for a standard, vendor-neutral format for 3D data, others may wonder about alternatives to the E57 format. Most 3D imaging system data exchange today takes place using one of three types of file formats: proprietary formats, ad-hoc formats, or the LAS format.

Proprietary formats are not an efficient approach to data exchange in the long term. Each hardware manufacturer and software company typically defines one or more proprietary formats for their data. As the industry grows and the number of options increases, there is a combinatorial explosion as each developer needs to create import and export capabilities for all other formats. Often, when transferring 3D data between different software packages, there is an existing, proprietary format that one program can export and the other can import. Occasionally, information is lost, reduced in accuracy, or corrupted due to programming errors or miscommunication between the writer and reader of the data. In situations where direct transfer using a common file format is not possible, conversion to a series of intermediate formats may be necessary. This workaround is both time-consuming and error prone, since there are more opportunities for data corruption or information loss. Finally, proprietary formats are not always openly available, and even when they are, they may not be well-documented or have a software library for reading or writing. The E57 format addresses these problems by providing a vendor-neutral format that, ideally, would eliminate the need for importing and exporting numerous proprietary formats.

In some cases, programmers invent their own ad-hoc formats for exchanging 3D information. Such formats are often limited to ASCII and are, consequently, not space or time efficient. Formal documentation of ad-hoc formats is usually limited or non-existent, which increases the chances for data corruption or misuse. Finally, these formats do not usually gain widespread usage, so the same concepts are continuously re-implemented in slightly different ways, ultimately leading to confusion and wasted effort. The E57 format reduces the need for ad-hoc formats, since there is an easy path to adoption through the libE57 software.

The LAS file format (short for LASer) is a data format designed to store 3D point cloud data that was developed by the ASPRS [4, 5]. It is specifically geared toward the needs of the aerial sensing community, though the format can be utilized for terrestrial laser scanner data by ignoring the inapplicable fields. The LAS format enjoys good support in the aerial sensing community and has recently begun to receive attention in the terrestrial laser scanning community. The E57 format is intended to be a more general format that is well-suited for storing data across a variety of application domains.

In the areas where the E57 format and LAS format capabilities overlap, the two formats take very different approaches to solving the same problem. For example, the LAS format uses fixed format records to specify information about each point. As new versions of the format are approved, new formats are added to the list of defined formats. The user must choose the format that is closest to the representation needs for a particular application. The E57 approach, on the other hand, allows users to flexibly choose the information associated with each 3D point as well as the number of bits used to represent the information. If the sensor only provides 10 bits of range information, it is possible to store just 10 bits for the range measurements.

The E57 format also includes some features that are not available in the current LAS format. The E57 format supports gridded data (i.e., data aligned in regular arrays), multiple coordinate systems (including Cartesian and spherical), embedded images from cameras, built in error detection, and groupings of points into rows, columns, or user-defined groups. The E57 format also defines an extension mechanism that allows users to develop custom capabilities that were not envisioned in the initial design of the standard. Finally, the E57 format has an essentially unlimited file size and number of records (18 exabytes, or $1.8x10^{19}$ bytes in length), whereas the LAS format is limited to 4.2 billion ($4.2x10^{9)}$ records. The data sets in the Digital Michelangelo project were up to 2 billion polygons in size, so even ten years ago, data sets were approaching these limits [6].

6. ONGOING AND FUTURE WORK

One of the goals of the E57 format design was to design and implement the core functionality of the format as quickly as feasible, while leaving some of the more challenging aspects of the design for future versions of the standard. This strategy has prevented the standard from becoming mired in controversy before it could ever be ratified. In ongoing work, we are developing a number of features and extensions that we expect to release as extensions or incorporate into the next version of the standard. Some of these capabilities include:

- *Advanced compression* – Compression of 3D data is a complex topic, and advanced work on compression was delayed until the next version of the standard. However, the potential benefits from improved compression are substantial because the data sets from existing sensors are already enormous and are getting larger with each generation of hardware.
- *Uncertainty representation* – Current 3D sensing representations rarely store information about the uncertainty of the data. However, this information is critical for making informed decisions in surveying and inspection applications. The challenge here is that there is no accepted representation for uncertainty, so it will require some work to determine a suitable representation and integrate it into the standard.
- *Mobile data support* – Many 3D imaging applications involve sensing from mobile platforms, such as cars or trains. In such cases, additional information must be kept in the 3D data file to track the sensor location at each time instant. We hope to incorporate the best practices from the mobile sensing community into an extension that will eventually be integrated into the standard itself.
- *Improved intensity representation* – The current standard supports the encoding of raw intensity values produced by a sensor. However, such a measure is imprecise, and even the term "intensity" is ambiguous. We are investigating methods for representing signal magnitude that are more accurately quantified so that data from different sensors could be combined in a useful manner. Ideally, one would like to encode properties of the underlying sensed surface, such as the surface reflectivity, but this is a difficult and potentially intractable problem.

7. ACKNOWLEDGEMENTS

The development of the E57 file format standard was a group effort by the entire ASTM E57.04 committee. The number of people that contributed substantially to this effort is too large to include each member as an author. However, the E57 standard would not exist without the enduring volunteer efforts of the committee members.

8. REFERENCES

[1] ASTM E57 Committee on 3D Imaging Systems - http://www.astm.org/COMMIT/COMMITTEE/E57.htm.

[2] Open Geospatial Consortium - http://www.opengeospatial.org.

[3] libE57: Software Tools for Managing E57 files - http://www.libe57.org/.

[4] A. Samberg, "An implementation of the ASPRS LAS standard," *IAPRS*, vol. XXXVI, 2007.

[5] The LAS file format - http://www.asprs.org/society/committees/standards/lidar_exchange_format.html.

[6] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk, "The Digital Michelangelo Project: 3D Scanning of Large Statues," in *Proceedings of ACM SIGGRAPH*, 2000, pp. 131-144.