ISSN 2464-4617 (print)
ISSN 2464-4625 (online)
Computer Science Research Notes - CSRN 3301
http://www.wscg.eu
WSCG 2023 Proceedings

Blocky Volume Package: a Web-friendly Volume Storage and Compression Solution

Žiga Lesar
University of Ljubljana
Faculty of Computer and Information Science
Večna pot 113
1000 Ljubljana, Slovenia
ziga.lesar@fri.uni-lj.si

Ciril Bohak
King Abdullah University of Science and Technology
Visual Computing Center
23955 Thuwal, Saudi Arabia
ciril.bohak@kaust.edu.sa

Matija Marolt
University of Ljubljana
Faculty of Computer and Information Science
Večna pot 113
1000 Ljubljana, Slovenia
matija.marolt@fri.uni-lj.si

ABSTRACT

The Blocky Volume Package (BVP) format is a distributed, platform-independent and API-independent format for storing static and temporal volumetric data. It is designed for efficient transfer over a network by supporting sparse volumes, multiple resolutions, random access, and streaming, as well as providing a strict framework for supporting a wide palette of encoding formats. The BVP format achieves this by dividing a volume or a volume sequence into blocks that can be compressed and reused. The metadata for the blocks are stored in separate files so that a client has all the information required for loading and decoding the blocks before the actual transmission, decoding and rendering take place. This design allows for random access and parallel loading and has been specifically designed for efficient use on the web platform by adhering to the current living standards. In the paper, we compare the BVP format with some of the most often implemented volume storage formats, and show that the BVP format supports most major features of these formats while at the same time being easily implementable and extensible.

Keywords

Volume storage, volume compression, block-based format.

1 INTRODUCTION

Volumetric data are a type of data that describe the properties or characteristics of a three-dimensional space. Such data are used to represent a wide range of physical phenomena, including but not limited to density, temperature, pressure, and composition of a particular region of space. Volumetric data play a significant role in various fields such as medical imaging, scientific visualization, and computer graphics.

Volumetric data can be represented in a number of different ways, depending on the specific application and the type of data being represented. Some common methods of representation include voxel grids, point clouds, and signed distance fields. In this paper, we will focus on storing volumetric data as voxel grids, which we will refer to as volumes. Volumes are a popular representation of volumetric data due to their simplicity, flexibility, and ease of use. They can be acquired through a variety of techniques, including computer simulations, crystallography, electron microscopy (transmission tomography [KM86], cryo-electron tomography [KK09]), computed tomography [KSKV90], positron emission tomography [BMTV05], magnetic resonance imaging [Fos84], and 3D ultrasound [NE93, HZ17]. These techniques allow for the creation of highly detailed and accurate data, making them a rich source of volumes for a wide range of applications. However, volumes also have a number of disadvantages that limit their potential applicability. One major disadvantage, which we address in this paper, is the high memory usage required for storage and manipulation, especially in applications where high resolution is required. Additionally, operations on volumes, such as rendering, filtering, segmentation, and registration, can be computationally expensive and time-consuming. In certain cases, alternative methods of representation, such as surface meshes or point clouds, may be more appropriate, but conversions to such representations inherently result in information loss, which is often unacceptable.

For volume storage, compression plays a vital role in managing large amounts of data by reducing storage requirements while retaining the quality of the

data [RGG+13, BRGIG+14]. In computer graphics, it alleviates the communication bottleneck problem, resulting in faster and more efficient visualizations, as well as reducing energy consumption which is critical on mobile devices. While storage and compression methods for two-dimensional images and videos have been subject to considerable research and innovation, which pushed the technology closer to theoretical limits, the technology for compressing and storing volumes has, in comparison, remained rather underdeveloped [SW03, BRLP20], despite the significantly higher storage requirements.

Existing volumetric data storage and compression formats have several shortcomings. The first issue is the lack of simple and effective compression solutions. Existing formats are either extremely simplistic and do not offer any compression capabilities, or they are featureful but difficult to understand and implement, making them less accessible to a wider range of users. Additionally, there is poor support for different platforms, as data types and endianness are often not specified, causing compatibility issues. Finally, poor extensibility is a major concern, as there is a lack of extension mechanisms, support for only a limited set of data formats, and a lack of support for GPU-accelerated formats. These problems highlight the need for better and more accessible solutions for volumetric data storage and compression.

As a potential solution to the listed shortcomings of the existing formats, this paper presents the Blocky Volume Package (BVP) format, a novel block-based format for distributed volume storage, compression and transmission. Compared with the existing formats, it includes a wide array of features and comes with a comprehensive list of advantages:

- it supports distributed volume storage, compression and transmission;
- it is agnostic to the underlying platform, storage medium, and graphics APIs;
- it is geared towards applications that require fast lookups and previews (e.g., rendering applications);
- it unifies concepts from several existing formats and generalizes them to create a more comprehensive and versatile format;
- it is simple to implement, integrate and extend.

2 RELATED WORK

There is a vast body of research on volumetric data storage and compression formats [RGG+13, LMG+18, LM14], covering a wide range of applications and use cases. Due to the scope and breadth of this field, we will focus specifically on formats for storing and compressing dense, bounded volumes. This includes formats such as DICOM [GPS05, MDG08, Clu21], and NIfTI[1], which are commonly used in medical imaging, as well as formats such as VTI [SML06, HMCA15], HDF5 [FHK+11, KR18], NRRD[2] and Zarr[3], which are used in scientific computing and other fields. While these formats may not be applicable to all types of volumetric data, they represent a significant portion of the volumetric data storage landscape and are relevant to a wide range of applications.

Volumes have historically been stored in raw format, which is a file format that stores data in its raw, unprocessed state, voxel by voxel. It is often used during data acquisition, where large amounts of data prohibit the use of on-the-fly compression. Because of this, raw format files are typically very large and can be difficult to work with. Raw format files do not include any information about the data type or endianness, which can make them incompatible with different platforms or software. Metadata for raw format files is usually stored separately (MHD[4], MRC [CHS96, CHM+15], NRRD, PVM [5], VFF[6], VOL[7]), and includes, among other information, the resolution of the volume, its orientation and data type. However, the lack of standardization makes such metadata formats a poor choice for interoperability. Despite these limitations, raw format is often used as an intermediary format and as a format for offline use. Some of the listed formats (MHD, MRC, NRRD, VFF) include basic compression support, which is applied directly to the raw data without any transformations or spatial data structures.

In recent years, several new formats have been developed that address some of the limitations of raw format. One such format is OpenVDB [MLJ+13, MAB19], which is a hierarchical representation of sparse volumetric data. OpenVDB uses a voxel grid data structure and a tree-based topology to efficiently store and manipulate sparse data. This allows for efficient data access and manipulation, and makes it well-suited for visual effects and simulation applications. However, it may not be well-suited for dense volumetric data, as it is optimized for sparse data, and can be significantly more difficult to implement compared to other common formats. NeuralVDB [KLM22, Cla22] is an extension of OpenVDB that leverages the hierarchical data structure of OpenVDB and enhances it with efficient deep neural network compression capabilities. This makes it possible to store and manipulate large volumes in a way that

[1] https://nifti.nimh.nih.gov
[2] https://teem.sourceforge.net/nrrd/
[3] https://zarr.dev
[4] https://itk.org/Wiki/ITK/MetaIO/Documentation
[5] http://paulbourke.net/dataformats/pvm/
[6] https://www.ventuz.com/support/help/latest/DevelopmentVFF.html
[7] http://paulbourke.net/dataformats/vol/

	RAW	BVP	MHD	NRRD	Zarr	VDB	MRC	NIfTI	DICOM	VTI	HDF5
Standard data types	no	yes	no	yes	yes	yes	no	yes	no	yes	yes
Extensible formats	no	yes	no	no	yes	yes	no	no	no	no	yes
Platform-independent	no	yes	yes	yes	yes	yes	no	no	yes	yes	yes
Simple to implement	yes	yes	yes	yes	no	no	yes	yes	no	no	no
Distributed storage	no	yes	no	yes	yes	yes	no	no	no	yes	yes
Storage alternatives	no	yes	no	no	yes	no	no	no	no	no	no
Format-agnostic operations	no	yes	no	yes	yes	no	no	no	no	no	no
Physical dimensions	no	yes	no	yes	no	no	yes	yes	yes	no	no
Extensions	no	yes	no	no	no	no	no	no	no	no	no
Multiresolution	no	yes	no	no	no	no	no	no	no	no	no
Animations	no	yes	no	yes	no	yes	no	no	yes	no	no
Supercompression	no	yes	no	yes	yes	yes	yes	yes	yes	yes	yes
GPU compression formats	no	yes	no	no	no	no	no	no	no	no	no
Higher-dimensional arrays	yes	no	no	yes	yes	no	no	no	no	no	yes
Sparse data	no	no	no	no	no	yes	no	no	no	no	yes
Transformations	no	no	no	yes	no	yes	no	yes	yes	no	no
General-purpose data	yes	no	no	yes	yes	no	no	no	no	no	yes

Table 1: Features supported in different formats.

is both memory-efficient and fast. Since NeuralVDB is built on top of OpenVDB, it shares many of its advantages as well as downsides, especially implementation complexity. NanoVDB [Mus21] is a lightweight GPU-accelerated version of OpenVDB, which primarily targets rendering applications.

VTI is the image format of the Visualization Toolkit (VTK). VTI is a general-purpose format that supports a wide variety of data types and primitives, including structured grids, unstructured grids, and polygons. As a result, it shares some of the downsides with other formats that are not optimized for volume storage. Newer versions of VTI support parallel I/O along with several compression options, including LZ4, LZMA, and ZLIB, enabling efficient reading and writing of large datasets. VTI supports only simple data formats, and GPU-accelerated compression formats are not supported. The data in a VTI file is described using an XML document, which can be complex to parse. VTI has the advantage of being widely supported and used in the scientific and research communities, and there are many libraries and tools available for working with VTI data. However, due to the broad feature set, implementing VTI can be a challenging task.

Another format that is gaining popularity is HDF5, which stands for Hierarchical Data Format version 5. HDF5 is a general-purpose data format designed to store large, complex data sets in a hierarchical format. It provides a rich feature set, including support for complex data structures, metadata, and parallel I/O. HDF5 is widely used and supported in many scientific and research communities, and has a wide range of libraries and tools available for working with HDF5 data. However, it is more complex and can be significantly more difficult to implement than other formats, and may not be as well-suited for storing volumes because the format is not aware of their specific spatial structure.

Zarr is a similar format as HDF5, it is also a general-purpose data format designed to store large, multi-dimensional arrays in a hierarchical structure. Zarr provides a simple, easy-to-use Python API, and can be used with a wide range of compression and encoding techniques. It has been designed to support distributed storage solutions, such as Amazon S3. However, it supports only a limited set of data formats, which does not include GPU-accelerated formats.

DICOM is a standard for storage, communication and management of medical images and related information, such as MRI and CT scans. DICOM was created to facilitate the exchange of information between different medical imaging devices and provide a standard format for storing and transmitting images and metadata. It is primarily a container format, and storage and processing information for each specific use case is provided in a separate specification document (e.g. there is a separate specification for 3D ultrasound images). DICOM is a large and complex standard that encompasses a wide range of use cases, with volume storage being only a small part of it. Unsurprisingly, implementing DICOM requires a significant investment of time and resources.

NIfTI is a file format for storing medical images and data, particularly in the field of neuroimaging. NIfTI was created as an alternative to the popular Analyze format and was designed to overcome some of the limitations of Analyze. NIfTI provides basic compression support through the use of DEFLATE, and it includes spatial transformation information, such as the orientation and size of the data. This information is stored in the header of the NIfTI file, along with information about the data type, data range, and measurement units. The use of physical units ensures that the data is correctly scaled and can be used for quantitative analysis. However, despite its advantages, NIFTI also has some limitations. The set of data formats is limited

and does not include GPU-accelerated compression formats. NIfTI is also not a hierarchical format, which prohibits efficient random access to the underlying data.

The BVP format addresses the downsides of the existing formats by unifying their advantages while following a simplistic design. One of the main advantages of BVP is its simplicity, which greatly facilitates integration and extension (e.g., as a plugin) of existing software. Unlike other formats, BVP additionally supports GPU-accelerated compression formats, which motivates its usage in rendering applications. For an overview of the features of different formats, refer to Table 1.

3 BLOCKY VOLUME PACKAGE

The main motivation behind BVP is its use on the web platform, where platform independence, memory safety, and efficient compression are crucial factors. The web presents additional restrictions, such as limited memory usage and restricted external file access. However, it simplifies some tasks, such as parsing JSON documents and manipulating arrays.

Drawing from the best features of the existing formats, we designed the BVP format with the following goals in mind:

- (G_1) the format must enable efficient parallel random access;
- (G_2) the format must support distributed storage and access;
- (G_3) metadata must be stored separately to enable clients to have all information about a volume available before the actual transmission, decoding and rendering take place;
- (G_4) data types have to support common use cases in a wide range of applications;
- (G_5) common GPU-accelerated compression formats, such as S3DC [YNV08], ETC [SAM05] and ASTC [NLP+12], must be supported;
- (G_6) the format must support compression in the form of reuse of parts of a volume or general-purpose compression;
- (G_7) execution of common operations, such as cropping and concatenation, must be possible without knowledge of the data format;
- (G_8) the format must be able to store multiple modalities, e.g. raw data and segmentation;
- (G_9) multiple resolutions of the same volume may optionally be added to allow fast previews;
- (G_{10}) physical units must be used to scale the volume correctly and enable quantitative analysis;
- (G_{11}) established data representation standards, such as IEEE 754, UTF-8 and JSON, must be respected throughout the format for it to be completely platform-independent; and
- (G_{12}) an extension mechanism must be available to allow future enhancements.

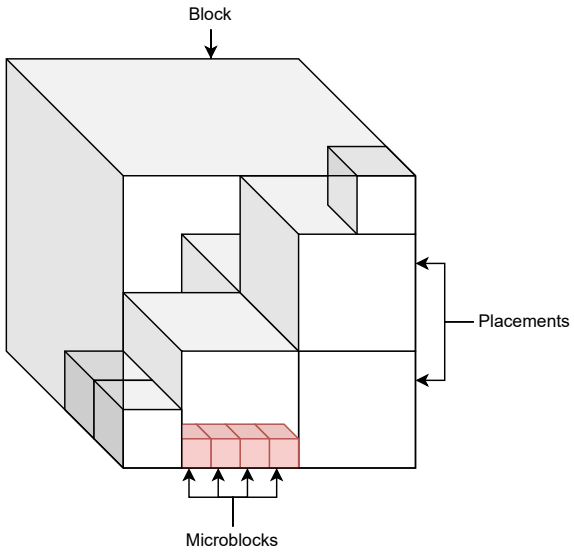

Figure 1: Blocks, microblocks and placements. A block is represented as a 3D array of microblocks. The microblocks come from external sources or other blocks copied into placements.

3.1 Overview

A BVP asset is built around 4 fundamental concepts: formats, microblocks, blocks, and modalities. A *microblock* is a fixed and indivisible block of voxels, represented by a specific number of bytes. The dimensions of a microblock, the number of bytes representing it, and the interpretation of these bytes are described by a *format*. Microblocks are assembled together to form a *block* as shown in Fig. 1. The data for a block may come from different sources, including external files and other blocks (G_2). Finally, blocks may be referenced by *modalities*, which equip a block with metadata and physical characteristics, such as its physical size. A BVP asset may contain multiple modalities such as raw data and different segmentations. Transformations are not described by a BVP asset, as it is only a storage format, not a scene description format.

All information about modalities, formats, blocks, how the blocks are structured in a hierarchy, and how to access the external microblock data, is supplied in a separate (G_3) JSON (G_{11}) document called the *manifest*.

3.2 Microblocks and formats

The concept of microblocks has been introduced to allow different block-level encoding schemes, such as S3DC, ETC, ASTC, etc., which are common in computer graphics and often hardware-accelerated (G_5). For example, a microblock may be as simple as a single voxel with a 32-bit floating-point value or more complex such as a $4 \times 4 \times 4$ RGBA volume represented by 16 bytes in ASTC format. There exist many encod-

ing schemes and choosing any fixed set of them for inclusion in BVP would make it inflexible and inextensible, as it would not be able to accommodate the diversity and evolution of encoding schemes. Instead, we chose to generalize the concept of a block-level encoding scheme as a format family, and a specific instance of that scheme as a format.

A format defines the dimensions of a microblock, the number of bytes representing it, and the semantics of these bytes when extracting voxel values. The dimensions and the number of bytes must be specified in a BVP asset so that even if an implementation encounters an unknown format (possibly added to BVP as an extension), it can still perform basic operations, such as block assembly, cropping and concatenation without having to decode a single microblock (G_7).

Formats that share similar characteristics are organized into format families, which can be added to BVP through extensions (G_{12}). The most basic format family is the "mono" format family, which describes single-voxel microblocks storing vectors of a single primitive type, for example one 8-bit unsigned integer or four 32-bit floating-point numbers. A format belonging to the "mono" format family must specify the primitive type and its size in bytes, along with the number of vector components. The primitive types are unsigned integers, signed integers represented as two's complements, or floating-point numbers conforming to the IEEE 754 standard, all of which are stored in little-endian format when endianness is relevant (G_{11}). Microblock byte size is the product of the number of components and the byte size of the primitive type. A conformant implementation must support at least vectors with 1 to 4 components, signed and unsigned integers represented with 1, 2, or 4 bytes, and single- and double-precision floating point numbers, which is adequate for most practical applications (G_4). Note that not all of these formats have equivalents in common graphics APIs, although they are commonly used in practice (e.g. doubles in scientific simulations and 16-bit integers in medical imaging), rationale being that BVP should not be limited by such APIs. In such cases, the data must be converted to an appropriate representation. It is important to note that data conversions are specific to pairs of formats and are therefore not included in the BVP specification.

3.3 Blocks

Blocks are cuboid regions of space formed by microblocks (see Fig. 1) of a specific format, which can be hierarchically assembled together to form volumes. Microblocks in a block adhere to a right-handed coordinate system and are ordered lexicographically in $\mathbb{R}^3$, which aligns with most modern graphics APIs. The data for a block can come from various sources such as files, web servers, or other blocks positioned in designated areas called placements within the parent block. This structure enables block reuse, resulting in a compressed volume (G_6), and additionally allows distributed storage (G_2).

An assembly process is required to fill a block with data. It is initialized with zeros, then placements are filled with referenced blocks, and finally, external data is copied into the block if available. Either a block has placements, or it is defined by external data. With future extensions in mind, we have decided not to explicitly disallow having both placements and external data present in a single block. This, however, necessitates an order of operations. Since the choice is largely arbitrary, we opted for placements first and external data second. The designated format of the parent block must be matched by all referenced sources.

External data can also be differential, in which case the existing microblocks are modified rather than overwritten, which can significantly improve compressibility. The placements must not be overlapping to allow parallel assembly (G_1). Additionally, circular dependencies between blocks are not allowed and an implementation should be robust to malformed assets. As a result, the blocks form a directed acyclic graph (DAG). Such a structure enables fast queries if an application chooses to store the blocks in unassembled form, or in out-of-core scenarios (G_1). The use of DAGs is not new in computer graphics [KSA13, DKB+16, vdLSE20], but to our knowledge, BVP is the first format to use DAGs for dense volume compression.

A block may reference a lower-resolution block representing the same data, which enables representing the same data in multiple resolutions, allowing fast previews and accelerated rendering (G_9). Additionally, rudimentary support for animated volumes is provided by allowing blocks to reference subsequent blocks as frames of an animation. Video compression is possible by sharing constituent blocks between frames. It is important to note that while this feature is available, the BVP format is not primarily designed to target volumetric video.

3.4 Modalities

Blocks are referenced by modalities which provide semantic information about the blocks, such as the physical size and acquisition method (G_{10}). A BVP asset can store multiple modalities, such as raw data and segmentation volumes, or even scans of the same subject using different methods like magnetic resonance imaging and computed tomography (G_8). This allows for a comprehensive representation of the data and enables the storage of multiple views and interpretations of the same underlying data. Unlike blocks in a single hierarchy, different modalities can use different formats.

3.5 Manifest

The manifest is a UTF-8-encoded JSON document (G_{11}), which may be split into several parts, that describes all the information stored in a BVP asset (G_3). It includes information about modalities, formats, blocks, the structure of blocks in a hierarchy, and how to access the external microblock data. This design draws inspiration from the glTF format. Furthermore, the manifest stores metadata about the BVP asset, such as copyright information, acquisition methods, checksums, and various timestamps.

External microblock data is referenced from the manifest by a URL (G_{11}), which may locate a file in a local file system, an archive, or a web server. Consequently, the BVP format is agnostic of the storage medium, at the same time allowing distributed storage and single-file volume archives (G_2). External data can optionally be supercompressed to further improve the compression ratio. A modified version of the LZ4 compression scheme, known for its simplicity and high decompression speeds, is a core component of the BVP format. Deflate, in contrast to LZ4, is much more commonly used and achieves better compression ratios at the cost of simplicity and decompression speed. For these reasons it is included in the BVP format as an extension.

3.6 Extension mechanism

The BVP format has a built-in extension mechanism, similar to the one used by the glTF format (G_{12}). This mechanism allows the core format to remain simple to implement, while still providing the ability to extend certain functionalities, for example by adding new format families, compression methods, and block placement capabilities. Extensions are categorized into two types: those that allow a BVP asset to be decoded even if the extension is not supported and those that require specific decoding logic, such as compression methods. Both types of extensions must be listed in the manifest so that they are readily available to the BVP reader.

The core BVP format, without any extensions, only describes the block assembly process, the "mono" format family, and modalities. A simplified LZ4 supercompression is also included. Everything else, such as compressed formats, multiple resolutions, animations, and differential blocks, is provided by extensions.

4 RESULTS

We compared the following existing formats in terms of file size: RAW (baseline), BVP, MRC, NRRD, VTI, NIfTI, Zarr, HDF5, OpenVDB, and DICOM. Since Zarr does not prescribe a storage medium, we used an uncompressed ZIP archive. Similarly, we used a simplified form of ASAR[8] for BVP assets. In the first set of tests we disabled supercompression and relied only on block reuse to reduce the file sizes. In the next set of tests, we enabled LZ4 supercompression where it was available. The dataset was comprised of 40 single-channel 8-bit integer volumes from various fields[9], adding up to approximately 1424 MB. As a practical use case, we also used BVP to store an ASTC-compressed RGBA volume to show that BVP can effectively handle GPU-accelerated compression formats. Finally, to show the potential for compression with DAGs, we evaluated BVP on a 32-bit instance segmentation volume. The dataset and the scripts used for storing it in different formats are available on github[10].

For the BVP assets we used a two-level block hierarchy with the parent block in the first level and subblocks of size $32 \times 32 \times 32$ in the second level. Equal blocks were found using the xxHash32 hash function[11].

Figure 2 reveals that BVP with block reuse outperforms other formats, which do not include this feature. This is most evident in volumes with lots of empty space. When we enabled LZ4 supercompression, we found that only the BVP format could produce a smaller file than LZ4 alone, as shown in Figure 3. Note that in Figures 2 and 3 we colored the baseline (the raw format without and with supercompression) yellow, the BVP format red, and other formats blue. Light blue was used in Figure 3 for the formats that are not directly comparable to BVP due to e.g. unsupported data types or unsupported compression algorithms. Some formats outperformed BVP, but they relied on other compression algorithms, such as DEFLATE (MRC, NRRD, NIfTI) or JPEG2000 (DICOM), as they do not support LZ4. OpenVDB does not support 8-bit integers and stores values as 32-bit floating-point numbers, which performs well without supercompression, but poorly with it. The only formats with comparable features were VTI, Zarr, and HDF5, which produced 4-8 % larger files and are significantly more complex in design and implementation. Additionally, HDF5 provides LZ4 only as a plugin and it does not allow arbitrary subblock sizes.

Compression ratios expectedly varied throughout the dataset depending on the amount of noise in the volumes. Where the amount of noise was reasonable (in around half of the volumes), we achieved reductions in file sizes from 10 % to 70 % without the use of LZ4 supercompression. With LZ4 supercompression, reductions varied more. However, when comparing LZ4-compressed raw volumes and LZ4-compressed BVP

[8] https://github.com/electron/asar

[9] https://klacansky.com/open-scivis-datasets/

[10] https://github.com/UL-FRI-LGM/wscg2023-bvp

[11] http://www.xxhash.com/

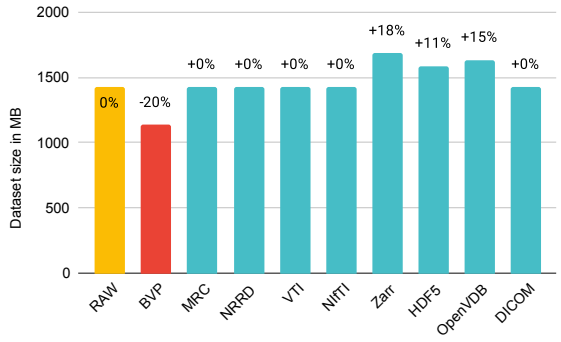

Figure 2: Dataset sizes without supercompression. The RAW baseline and BVP were colored yellow and red, respectively, for emphasis.

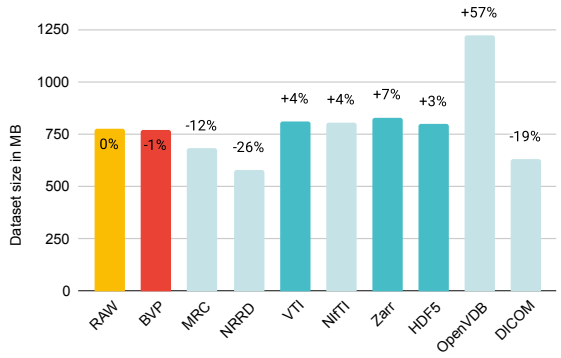

Figure 3: Dataset sizes with supercompression. The RAW baseline is a LZ4-compressed RAW dataset. Light blue formats are not directly comparable to BVP.

volumes, the resulting file sizes were very similar, with differences of up to 5 %.

Smaller blocks could be used to improve the compression ratio at the expense of a bigger manifest and a lower compression speed. However, due to diminishing returns, we found that blocks of size $32 \times 32 \times 32$ were a good compromise.

Note that these results are for lossless compression only. Extending the condition for block reuse from equality to similarity would result in lossy compression, which would significantly increase the compressibility of the volumes due to the use of DAGs. Even though this feature has not yet been implemented in our compressor, the BVP format readily supports it. Similarly, differential blocks were not included in the evaluation. We conjecture that the benefit would be marginal in the static case and more pronounced in animations.

In addition to raw volume compression, we evaluated BVP on an ASTC-compressed RGBA volume of size $768 \times 768 \times 360$. We used microblocks of dimensions $4 \times 4 \times 4$ and $6 \times 6 \times 6$ and a similar two-level block hierarchy with subblocks of size $48 \times 48 \times 48$. Both cases were evaluated with and without LZ4 supercompression. Figure 4 shows that BVP with LZ4 achieves

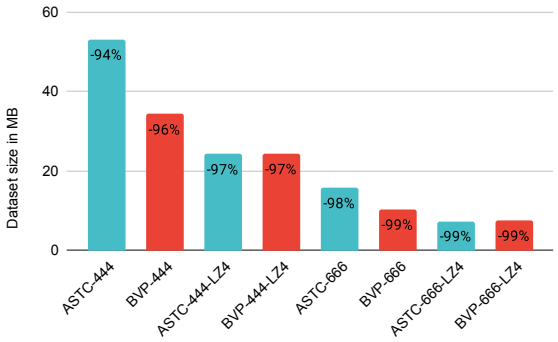

Figure 4: Dataset size using ASTC compression. Percentages are calculated with respect to the RAW file size (approximately 850 MB).

approximately equal final asset size as raw ASTC with LZ4 supercompression, but with added metadata and hierarchical structure.

To show the compression capabilities of DAGs, we used BVP to compress a 32-bit instance segmentation volume of size $512 \times 512 \times 512$. We used the mono format and a two-level block hierarchy with subblocks of size $32 \times 32 \times 32$. The raw volume of 512 MB was reduced to 259 MB, which corresponds to a compression ratio of 50 %. After enabling LZ4 compression, the raw volume was reduced to 6.1 MB, while the BVP-compressed volume was reduced to 6.9 MB. The overhead was almost exclusively due to the manifest.

5 CONCLUSION

The BVP format offers many advantages for storing and distributing volumes. It is a simple-to-implement platform-independent format that can be used to store large volumes from a variety of different fields. BVP draws inspiration from many existing formats, consolidating ideas such as multiresolution representation, hierarchical storage, parallel I/O, and supercompression. It readily supports GPU-accelerated compressed formats for efficient use in computer graphics. Contrary to the existing formats, the BVP format respects the widely-accepted engineering standards, facilitating data exchange and interoperability.

However, there are some limitations to BVP. BVP by design only supports dense, bounded 3D volumes. Specifically, it does not support unlimited indexing (such as in OpenVDB), it does not support general-purpose data storage (such as in HDF5), and it does not support higher-dimensional arrays (such as in NRRD and Zarr). Additionally, a BVP asset does not include information about the scene, such as transformations, transfer functions, and illumination, so a separate format, such as USD[12] [CMMLB22] or

[12] https://graphics.pixar.com/usd/

glTF [RAPC14], must be used for that purpose. Furthermore, BVP deliberately excludes information about sampling and data conversions, as these operations are format-specific and often domain-specific.

There are also some technical limitations that have been deliberately introduced to simplify the implementation process. BVP has some restrictions on block and placement dimensions and formats. For instance, blocks and placements must be completely contained within the parent block, and the size of a block and placement must be an integer multiple of microblock size. Consequently, block-level encoding schemes prohibit arbitrary block sizes. Additionally, the format of placements must match the format of the block, and format mixing is only allowed on modality level.

Despite the above limitations, the BVP format provides a simple and comprehensive representation of volumes from various fields. It is a reasonable and featureful alternative to the existing formats.

6 REFERENCES

[BMTV05] Dale L Bailey, Michael N Maisey, David W Townsend, and Peter E Valk. *Positron emission tomography*, volume 2. Springer, 2005.

[BRGIG+14] M. Balsa Rodríguez, E. Gobbetti, J.A. Iglesias Guitián, M. Makhinya, F. Marton, R. Pajarola, and S.K. Suter. State-of-the-art in compressed gpu-based direct volume rendering. *Computer Graphics Forum*, 33(6):77–100, 2014.

[BRLP20] Rafael Ballester-Ripoll, Peter Lindstrom, and Renato Pajarola. Tthresh: Tensor compression for multidimensional visual data. *IEEE Transactions on Visualization and Computer Graphics*, 26(9):2891–2903, 2020.

[CHM+15] Anchi Cheng, Richard Henderson, David Mastronarde, Steven J. Ludtke, Remco H.M. Schoenmakers, Judith Short, Roberto Marabini, Sargis Dallakyan, David Agard, and Martyn Winn. Mrc2014: Extensions to the mrc format header for electron cryo-microscopy and tomography. *Journal of Structural Biology*, 192(2):146–150, 2015.

[CHS96] R.A. Crowther, R. Henderson, and J.M. Smith. Mrc image processing programs. *Journal of Structural Biology*, 116(1):9–16, 1996.

[Cla22] Ronald Clark. Volumetric bundle adjustment for online photorealistic scene capture. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6124–6132, June 2022.

[Clu21] David A. Clunie. Dicom format and protocol standardization—a core requirement for digital pathology success. *Toxicologic Pathology*, 49(4):738–749, 2021.

[CMMLB22] Jon-Patrick Collins, Romain Maurer, Fabrice Macagno, and Christian Lopez Barron. Usd at scale. In *The Digital Production Symposium*, DigiPro '22. Association for Computing Machinery, 2022.

[DKB+16] Bas Dado, Timothy R. Kol, Pablo Bauszat, Jean-Marc Thiery, and Elmar Eisemann. Geometry and attribute compression for voxel scenes. *Computer Graphics Forum*, 35:397–407, 5 2016.

[FHK+11] Mike Folk, Gerd Heber, Quincey Koziol, Elena Pourmal, and Dana Robinson. An overview of the hdf5 technology suite and its applications. page 36–47, New York, NY, USA, 2011. Association for Computing Machinery.

[Fos84] Margaret A Foster. Magnetic resonance in medicine and biology. *Radiology and Nuclear Medicine*, 1984.

[GPS05] R.N.J. Graham, R.W. Perriss, and A.F. Scarsbrook. Dicom demystified: A review of digital file formats and their use in radiological practice. *Clinical Radiology*, 60(11):1133–1140, 2005.

[HMCA15] Marcus D. Hanwell, Kenneth M. Martin, Aashish Chaudhary, and Lisa S. Avila. The visualization toolkit (vtk): Rewriting the rendering code for modern graphics cards. *SoftwareX*, 1-2:9–12, 2015.

[HZ17] Qinghua Huang and Zhaozheng Zeng. A review on real-time 3d ultrasound imaging technology. *BioMed research international*, 2017, 2017.

[KK09] Roman I Koning and Abraham J Koster. Cryo-electron tomography in biology and medicine. *Annals of Anatomy-Anatomischer Anzeiger*, 191(5):427–445, 2009.

[KLM22] Doyub Kim, Minjae Lee, and Ken Museth. Neuralvdb: High-resolution sparse volume representation using hierarchical neural networks, 2022.

[KM86] Satoshi Kawata and Shigeo Minami. The principle and applications of optical microscope tomography. *Acta his-*

tochemica et cytochemica, 19(1):73–81, 1986.

[KR18] Quincey Koziol and Dana Robinson. Hdf5. [Computer Software] https://doi.org/10.11578/dc.20180330.1, mar 2018.

[KSA13] Viktor Kämpe, Erik Sintorn, and Ulf Assarsson. High resolution sparse voxel dags. *ACM Transactions on Graphics*, 32:1–13, 7 2013.

[KSKV90] Willi A Kalender, Wolfgang Seissler, Ernst Klotz, and Peter Vock. Spiral volumetric CT with single-breath-hold technique, continuous transport, and continuous scanner rotation. *Radiology*, 176(1):181–183, 1990.

[LM14] Michele Larobina and Loredana Murino. Medical image file formats. *Journal of Digital Imaging*, 27(2):200–206, 2014.

[LMG+18] S. Li, N. Marsaglia, C. Garth, J. Woodring, J. Clyne, and H. Childs. Data reduction techniques for simulation, visualization and data analysis. *Computer Graphics Forum*, 37(6):422–447, 2018.

[MAB19] Ken Museth, Nick Avramoussis, and Dan Bailey. Openvdb. In *ACM SIGGRAPH 2019 Courses*, SIGGRAPH '19, New York, NY, USA, 2019. Association for Computing Machinery.

[MDG08] Mario Mustra, Kresimir Delac, and Mislav Grgic. Overview of the dicom standard. In *2008 50th International Symposium ELMAR*, volume 1, pages 39–44, 2008.

[MLJ+13] Ken Museth, Jeff Lait, John Johanson, Jeff Budsberg, Ron Henderson, Mihai Alden, Peter Cucka, David Hill, and Andrew Pearce. Openvdb: An open-source data structure and toolkit for high-resolution volumes. In *ACM SIGGRAPH 2013 Courses*, SIGGRAPH '13, New York, NY, USA, 2013. Association for Computing Machinery.

[Mus21] Ken Museth. Nanovdb: A gpu-friendly and portable vdb data structure for real-time rendering and simulation. In *ACM SIGGRAPH 2021 Talks*, SIGGRAPH '21, New York, NY, USA, 2021. Association for Computing Machinery.

[NE93] T.R. Nelson and T.T. Elvins. Visualization of 3d ultrasound data. *IEEE Computer Graphics and Applications*, 13(6):50–57, 1993.

[NLP+12] Jorn Nystad, Anders Lassen, Andy Pomianowski, Sean Ellis, and Tom Olson. Adaptive scalable texture compression. In *Proceedings of the Fourth ACM SIGGRAPH / Eurographics Conference on High-Performance Graphics*, EGGH-HPG'12, page 105–114, Goslar, DEU, 2012. Eurographics Association.

[RAPC14] Fabrice Robinet, Rémi Arnaud, Tony Parisi, and Patrick Cozzi. gltf: Designing an open-standard runtime asset format. 5:375–392, 2014.

[RGG+13] Marcos Balsa Rodríguez, Enrico Gobbetti, José A. Iglesias Guitián, Maxim Makhinya, Fabio Marton, Renato Pajarola, and Susanne K. Suter. A Survey of Compressed GPU-Based Direct Volume Rendering. In M. Sbert and L. Szirmay-Kalos, editors, *Eurographics 2013 - State of the Art Reports*. The Eurographics Association, 2013.

[SAM05] Jacob Ström and Tomas Akenine-Möller. Ipackman: High-quality, low-complexity texture compression for mobile phones. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, HWWS '05, page 63–70, New York, NY, USA, 2005. Association for Computing Machinery.

[SML06] Will Schroeder, Ken Martin, and Bill Lorensen. *The visualization toolkit an object-oriented approach to 3D graphics, 4th Edition*. Kitware, 2006.

[SW03] J. Schneider and R. Westermann. Compression domain volume rendering. In *IEEE Visualization, 2003. VIS 2003.*, pages 293–300, 2003.

[vdLSE20] Remi van der Laan, Leonardo Scandolo, and Elmar Eisemann. Lossy geometry compression for high resolution voxel scenes. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 3:1–13, 4 2020.

[YNV08] Héctor Yela, Isabel Navazo, and Pere-Pau Vazquez. S3Dc: A 3Dc-based Volume Compression Algorithm. In Luis Matey and Juan Carlos Torres, editors, *CEIG 08 - Congreso Espanol de Informatica Grafica*. The Eurographics Association, 2008.