Applications of the VOLA Format for 3D Data Knowledge Discovery

Conference Paper · July 2017

7 authors, including:

Jonathan Byrne
Intel
29 PUBLICATIONS 212 CITATIONS

Sam Caulfield
Intel
9 PUBLICATIONS 9 CITATIONS

Léonie Buckley
Trinity College Dublin
5 PUBLICATIONS 2 CITATIONS

Xiaofan Xu
Trinity College Dublin
7 PUBLICATIONS 9 CITATIONS

Some of the authors of this publication are also working on these related projects:

- EoT (Eyes of Things) View project
- Interest-point-based human body tracking View project

All content following this page was uploaded by Jonathan Byrne on 20 September 2017.
The user has requested enhancement of the downloaded file.
Applications of the VOLA Format for 3D Data Knowledge Discovery.

Jonathan Byrne, Sam Caulfield, Léonie Buckley, Xiaofan Xu, Dexmont Pena, Gary Baugh and David Moloney
Computer Vision and Machine Learning Group, Movidius / Intel
Email: jonathan.byrne@intel.com, sam.caulfield@intel.com, leonie.buckley@intel.com, xu.xiaofan@intel.com,
dexmont.pena@intel.com, gary.baugh@intel.com, david.moloney@intel.com

Abstract—VOLA is a compact data structure that unifies computer vision and 3D rendering and allows for the rapid calculation of connected components, per-voxel census/accounting, CNN inference, path planning and obstacle avoidance. Using a hierarchical bit array format allows it to run efficiently on embedded systems and maximize the level of data compression. The proposed format allows massive scale volumetric data to be used in embedded applications where it would be inconceivable to utilize point-clouds due to memory constraints. Furthermore, geographical and qualitative data is embedded in the file structure to allow it to be used in place of standard point cloud formats. This work examines the reduction in file size when encoding 3D data using the VOLA format and finds that it is an order of magnitude smaller than the current binary standard for point cloud data.

Index Terms—Point Clouds, Embedded Systems, Auralisation, Deep Learning, GIS.

I. INTRODUCTION

The worlds of computer vision and graphics, although separate, are slowly being merged in the field of robotics. Computer vision is taking input from systems, such as light detection and ranging (LIDAR), structured light or camera systems and generating point clouds or depth maps of the environment. This data then must be represented internally for the environment to be interpreted correctly. Unfortunately the amount of data generated by modern sensors quickly becomes too large for embedded systems. An example of the amount of memory required by dense representations is SLAMbench (kFusion) which requires 512 MiB to represent a 5 m³ volume with 1 cm accuracy [1]. A terrestrial LIDAR scanner generates a million unique points per second [2] and an hour long aerial survey can generate upwards of a billion unique points.

The result of having such vast quantities of data is that it soon becomes impossible to process, let alone visualize the data on all but the most powerful systems. Consequently it is almost never used directly. It is simplified by decimation, flattened into a DEM, or meshed using a technique such as Delaunay triangulation or Poisson reconstruction.

The original intention of VOLA was to develop a format that enabled an embedded system to process point cloud data into an internalized 3D model to easily navigate the environment. It has since been applied to numerous applications, such as ray-casting, deep learning, auralisation, machine learning and 3D Mapping. VOLA has shown itself to be a compact yet versatile format. This paper gives a summary of the different applications areas and the results obtained.

II. RELATED RESEARCH

There exist several techniques for organizing point cloud data and converting it to a solid geometry. Point clouds are essentially a list of coordinates, with each line containing positional information as well as colour, intensity, number of returns and other attributes. Although the list can be sorted using the coordinate values, normally a spatial partitioning algorithm is applied to facilitate searching and sorting the data. Commonly used approaches are the the Octree [3] and the KD-Tree [4].

Octrees are based in three dimensional space and so they naturally lend themselves to 3D visualization. There are examples where the Octree itself is used for visualizing 3D data, such as the Octomap [5]. Octrees are normally composed of pointers to locations in memory which makes it difficult to save the structure as a binary. One notable exception is the DMGOctree [6] which uses a binary encoding for the position of the point in the octree. Three bits are used as a unique identifier for each level of the Octree. The DMGOctree uses a 32 or 64 bit encoding for each point to indicate the location in the tree to a depth of 10 or 20 respectively.

Another technique for solidifying and simplifying a point cloud is to generate a surface that encloses the points. A commonly used approach that locally fits triangles to a set of points is Delaunay triangulation [7]. It maximizes the minimum angle for all angles in the triangulation. Triangular irregular networks (TIN) [8] are extensively used in Geographical Information Systems (GIS) and are based on Delaunay triangulation. One issue with that approach is that noise and overlapping points can cause the algorithm to make spurious surfaces.

A more modern and accurate meshing algorithm is Poisson surface reconstruction [9]. The space is hierarchically partitioned and information on the orientation of the points is used to generate a 3D model. It has been shown to generate accurate models and it is able to handle noise due to the combination of global and local point information. Poisson reconstruction will always output a watertight mesh but this can be problematic when there are gaps in the data. In an attempt to fill areas with little information, assumptions are made about the shape which can lead to significant distortions. There are also problems with small, pointed surface features which tend to be rounded off or removed by the meshing algorithm.

Finally there are volumetric techniques encoding point clouds. Traditionally used for rasterizing 3D data for render-
ing [10], the data is fitted to a 3D grid and occupancy of a point is represented using a volumetric element, or “Voxel”. Voxels allow the data to be quickly searched and traversed due to being fitted to a grid. While it simplifies the data and may merge many points into a single voxel, each point will have a representative voxel. Unlike meshing algorithms, voxels will not leave out features but conversely may be more sensitive to noise. The primary issue with voxel representations is that they encode everything, including open space. This means that there is a cubic relationship between the number of voxels that need to be stored and the volume resolution. For example, if the resolution is doubled then the memory requirements increase by a factor of 8.

VOLA combines the hierarchical structure of the Octree with traditional volumetric approaches, enabling it to only encode for occupied voxels. The approach is described in detail below.

III. THE VOLA FORMAT

VOLA is unique in that it combines the benefits of partitioning algorithms with a minimal voxel format. It hierarchically encodes 3D data using modular arithmetic and bit counting operations applied to a bit array. The simplicity of this approach means that it is highly compact and can be run on hardware with simple instruction sets. The choice of a 64 bit integer as the minimum unit of computation means that modern processor operations are already optimized to handle the format. While octree formats either need to be fully dense to be serialized or require bespoke serialization code for sparse data, the VOLA bit array is immediately readable without header information.

VOLA is built on the concept of hierarchically defining occupied space using “one bit per voxel” within a standard unsigned 64 bit integer. The one-dimensional bit array that makes up the integer value is mapped to three-dimensional space using modular arithmetic. The bounding box containing the points is divided into 64 cells and if there are points contained within a cell the bit is then set to 1 otherwise it is set to zero. The result of the first division is shown in Figure 1.

For the next level each occupied cell is assigned and additional 64 bit integer and the space is further subdivided into 64 cells. Any unoccupied cells on the upper levels are ignored allowing each 64 bit integer to only encode for occupied space. The bits are again set based on occupancy and appended to the bit array. The number of integers in each

<table>
<thead>
<tr>
<th>Implementation</th>
<th>VOLA</th>
<th>Octree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traversal Arithmetic</td>
<td>Bit Sequence</td>
<td>Pointer Based</td>
</tr>
<tr>
<td>Variable Depth</td>
<td>Modular</td>
<td>Pointer</td>
</tr>
<tr>
<td>Dense Search Complexity</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Sparse search complexity</td>
<td>O(1)</td>
<td>O(h)</td>
</tr>
<tr>
<td>Embedded System Support</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Look Up Table (LUT) Support</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Easily Save to File</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>File Structure</td>
<td>Implicit</td>
<td>Explicit</td>
</tr>
<tr>
<td>Cacheable</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Hierarchical Memory Structure</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

TABLE I: A comparison between VOLA and Octrees.

Fig. 1: Tree depth one, the space is subdivided into 64 cells. The occupied cells are shown in green.

Fig. 2: Tree depth two. Each occupied cell is subdivided into 64 smaller cells.
Fig. 3: Tree depth three. The process is repeated until a sufficiently granular resolution is obtained.

Fig. 4: Depth 1.

Fig. 5: Depth 2.

Fig. 6: Depth 3.

Fig. 7: Depth 4.

level can be computed by summing the number of occupied bits in the previous level. The resolution increases fourfold for each additional level as shown in figure 2.

The process is repeated with each level increasing the resolution of the representation by four until a resolution suitable for the data is reached. This depends on the resolution or points per meter of the data itself.

A. Two Bits Per Voxel

While the basic VOLA format can give occupancy information in a 3D environment, it cannot store information about the voxels. A backward compatible variation of the format was adopted in order to add this functionality. The VOLA Model is essentially mirrored so that each 64 bit block has an additional 64 bit block appended to it. The additional block contains the information related to that volume. The information resolution is less than the voxel resolution but the additional overhead only equates to an additional bit per voxel. The result is that
the file size doubles but as it is already a fraction of the size of the original point cloud, the difference is negligible. The 64 bits can be adapted to contain a variety of payloads, such as material properties, geographical information and colour information.

This section has described the basic implementation of the VOLA format and the adapted format for embedding information about the voxels. The following sections describe the applications that have been built upon this work.

IV. AUDIO

Auralisation is used to describe rendering audible (imaginary) sound fields[11]. For the purpose of JIT (Just In Time) auralisation, the environment being modelled is represented using VOLA. The response being modelled depends on two main factors - how far the sound travels between source and observer, and by how much it is attenuated.

The distance travelled indicates how much the response will be delayed with respect to the source, and the attenuation indicates how the power of the sound source is diminished. The attenuation is determined by the medium that the sound rays travel through/reflects off (e.g. through air or off a wall). Both the distance travelled and the attenuation of the sound can be obtained from the VOLA model.

Using Pythagoras’ theorem, the distance between the sound source and an occupied voxel is calculated. This is repeated for the return distance between the same voxel and the observer. These two distances are then combined as the total distance travelled by the sound ray and is known as an early reflector.

Currently only direct sound and early reflectors are taken into account as they contain the majority of the sound energy of the acoustic response. These rays and their energies are shown in figures 8a and 8b respectively. Direct sound rays are sound rays that travel directly to the observer without reflecting off any surfaces/objects.

The reflection coefficients are stored in the second bit of VOLA’s two per voxel representation, as described in section III-A. This can be manually set for an entire scene (e.g. if it is known that all the walls and floor are wooden). Manually measuring the reflection coefficient of materials is a complex process[12], however, CNNs can be trained to distinguish simple materials[13] (e.g. wood, metal, carpet), and assign predetermined reflection coefficients to voxels of these materials.

For optimisation, voxels that cause the same attenuation due to distance travelled and reflection coefficients are binned together. This is referred to as ”thresholding” the environment. Each of these bins are combined to generate an FIR filter through which the sound source is passed. The process of constructing this filter is shown in figure 9. The response from multiple sound sources can be calculated simultaneously through generating multiple filters.

An algorithm, SonifEye[14], implementing auralisation as described above is implemented on the Myriad 2[15]. The sounds source can be pre-recorded e.g. a musical recording, or can be recorded using a microphone. The runtime for auralising a single sound source in a 256 voxel cubed environment is given in table II.

The user’s surrounding are often neglected when providing audio for VR. SonifEye allows for more realistic VR audio to be generated in real time, eliminating the need to store canned audio.

V. DEEP LEARNING AND CONVOLUTIONAL NEURAL NETS.

Performing object recognition on 3D point-cloud occluded volumes depicting real-world scenes containing ubiquitous objects is an important problem in the computer vision field. It

<table>
<thead>
<tr>
<th>Step</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threshold Environment</td>
<td>8.20</td>
</tr>
<tr>
<td>Create Filter</td>
<td>0.3</td>
</tr>
<tr>
<td>Apply Filter</td>
<td>4.05</td>
</tr>
</tbody>
</table>

TABLE II: SonifEye Step Timings for a 256 voxel cubed environment for a sound source 1 second is length
present multiple challenges, including the significant memory requirements for these volumetric representations as well as real-time recognition of objects in mobile, power-constrained or autonomous conditions.

Convolutional Neural Networks (CNNs) have been shown to be powerful classification tools for multiple real-world computer vision tasks, such as scene segmentation and labelling[16], medical imaging diagnosis[17], [18] and object detection in autonomous driving[19], in many cases approaching near-human performance. As CNNs take on new and more challenging recognition tasks, they have been increasing in complexity, thus requiring larger datasets for training. The lack of readily available training data and memory requirements are two of the factors hindering the training and accuracy performance of 3D CNNs. Based on the previous works [20] [21], a voxelised point-cloud dataset was created containing 10 different 3D objects associated with different scenes in VOLA format shown in Fig.10. This dataset can minimize the memory footprint as well as increase the efficiency of CNN performance. In order to cope with high computational power with low-power supply requirements and low-energy consumption levels for real-time applications, the CNN model trained with our dataset is ported to a very low power and low cost Fathom NCS [22] based on Myriad2 MA2450 VPU [15] and can perform inference in 11 ms.

In order to train the network on our 3D objects with scene, we have developed a new architecture shown in Fig. 11. The input to the network is $64 \times 64 \times 64$ VOLA format object with scene shown in Fig.10. There are 3 convolutional layers, 1 fully connected layer and a softmax layer to produce the output. We also add a dropout layer after the third convolutional layer with drop rate of 0.5. The first convolutional layer produces 8 feature blocks with size $57^3$, the second and the third convolutional layer produces 16 feature blocks with size $50^3$ and $43^3$ respectively. After that we add 1 fully connected layer to produce 10 units and the final output gives the probability for different classes based on the input. This 3D CNN model is designed and trained using Caffe [24]. The accuracy achieved for the 3D CNN is shown in Table III.

VI. KFUSION

Simultaneous localisation and mapping (SLAM) is a classic computer vision problem which requires a significant amount of computational resource. Most SLAM solutions can be categorized as either a dense or sparse approach. For sparse techniques only a subset of image/sensor features are tracked in order to map an environment and localize the position and orientation of the sensor. However for dense approaches the main goal is to create a compact map with sufficient detail of the environment being captured.

KFusion[28] is a dense SLAM open source implementation of Microsoft’s KinectFusion[29] algorithm. KFusion has been ported to the Myriad 2 platform with the goal of using VOLA in the pipeline for storing compact volumetric 3D data. By using VOLA, large maps constructed by the algorithm will occupy a smaller memory footprint compared to the default dense voxel grid of truncated signed distance function and weights. Figure 12 shows the hardware setup for a live KFusion implementation running on Movidius’ Myriad 2 platform. A demonstration of KFusion running on the Myriad chip is available online at [30].

VII. TERRESTRIAL SCALE POINT CLOUD MAPPING IN REAL TIME.

Laser scan (LIDAR) surveys have been carried out over most major cities due to the high accuracy and speed of data collection. Furthermore the open data movement has meant that many governments have made this data publicly available for public use. Unfortunately the size of the point clouds generated from the scans is so large (10 gigabytes up to a terabyte) that it is almost never used directly. Converting it to the VOLA format allows for the size of the data to

![Fig. 11: 3D CNN architecture layout for Objects with background. The input is a $64 \times 64 \times 64$ object in VOLA format with each voxel represented by a single bit. The input passes through 3 convolutional layers and 1 fully connected layer. The kernels used for convolutional layers are $8 \times 8 \times 8 \times 8$ with stride 1 which gives the output feature blocks with size $57^3$, $50^3$ and $43^3$ from Conv1, Conv2 and Conv3 respectively. The output from fully connected layer is 10 units.](image)

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>PointNet</td>
<td>77.6%</td>
</tr>
<tr>
<td>3D ShapeNet</td>
<td>84%</td>
</tr>
<tr>
<td>OctNet</td>
<td>90.1%</td>
</tr>
<tr>
<td>Our 3D CNN</td>
<td>91.3%</td>
</tr>
<tr>
<td>VoxNet</td>
<td>92%</td>
</tr>
</tbody>
</table>

TABLE III: Accuracy achieved for different existing CNN models and our model. The existing CNN models are trained and tested on pure ModelNet10 objects where our 3D CNN is trained and tested on voxelized point-clouds contain ModelNet10 objects with background. (Higher percentage indicates better performance).

![Fig. 10: ModelNet10 [23] objects placed in different rooms with voxelised representation in VOLA format.](image)
be massively reduced while still allowing it to be used for mapping and machine learning applications.

The format specification has been designed to be compatible with terrestrial mapping applications. The point cloud data is georeferenced using both the WGS84 coordinate system [31] and the projected coordinate system that it was originally recorded in. This allows for all data to be georeferenced while minimizing distortion due to re-projection.

In order for the format to be used with standard GIS applications, an approach was developed to embed information about the voxels while minimizing memory usage. Semantic information, such as color, classification, number of returns and height, is embedded in the data using the two bits per voxel technique. An example of this is given in figures 13 and 14 which show the height data and the number of returns from the LIDAR scanner respectively.

VIII. EMBEDDED RAYCASTING

In autonomous applications, a compact and highly-detailed map is not useful unless the autonomous agent can meaningfully interact with it. One of the most fundamental interactions is collision detection. Accurate and high-performance collision detection is of particularly high importance in applications such as drones, where collision at speed can result in damage to property and harm to individuals. Common methods of ray/geometry intersection involve spatial partitioning schemes such as the KD-tree or octree as mentioned previously. The octree is the most popular technique for geometry arranged in regular grids. I compares the VOLA format to octrees in a number of scenarios. However, for certain architectures, the VOLA format provides other significant improvements over the octree.

A. Parallelism

In a parallel program, cores may have to wait longer on average to perform writes and removes to a pointer-based octree when compared to VOLA. This is because of the cascading memory allocations and memory frees that would have to be performed during the critical section. In the worst case VOLA would only require a cascade of memory writes, resulting in less time spent waiting on unlocks.

B. Cache-friendliness

A level in an octree uses pointers to indicate the occupancy of the level below. On a 64-bit system, the pointer uses 8 bytes of memory where a single bit would suffice to represent occupancy. The ratio of metadata (pointers) to data (voxels) is far lower in VOLA because everything is a voxel. This results in better cache friendliness because voxel data does not compete with pointer data for space in the cache.

C. Level of detail

VOLA supports similar level of detail to that of octrees. In an octree, the number of voxels in the first level is 8 and this number increases by a factor of 8 each successive level. In VOLA, this number is simply 64. This means the VOLA levels of detail converge on the full level of detail faster while still offering several degrees of coarseness.

D. Data locality

Using heap-allocation for the octree can result in highly fragmented memory and no assumptions can be made about
the relative positions of data in memory. In VOLA, the layout of the voxels is totally deterministic and very compact. Additionally, VOLA does not lose the ability to use Morton Encoding[32] to enhance data locality. In situation where multiple VOLA structures are used in parallel (e.g. to store subsampled colour information in addition to voxels), a relative address jump can be made from one VOLA structure to the same point in the other.

E. Efficient Ray-casting

The above characteristics of VOLA result in high performance ray-casting on amenable platforms. Efficient algorithms exist for ray/grid intersection tests[33]. These algorithms can be further optimised to make use of level of detail to skip a large number of tests that would otherwise test known-empty space. Multi core systems can use the extra cores to cast multiple rays in parallel.

The Movidius Myriad 2 is a heterogenous VPU that features 12 vector processors and an on-chip addressable cache memory. Each vector processor has a 128 KiB preferential slice of the 2 MiB on-chip memory which facilitates fast low-power access. Using VOLA, each slice can comfortably store its own copy of the top three levels of detail of the VOLA data structure. The remaining levels of the structure are stored in DDR memory.

A realtime raycaster was implemented on the Movidius Fathom stick as an example of the raycasting performance. The rays produce a depth image which visualises the voxels directly without producing an intermediate data structure such as a triangular mesh.

IX. Conclusions

VOLA has proven to be a versatile and compact format. It has been used for applications as diverse as audio modelling, machine learning, and storing georeferenced geo-data. It allows for massive compression of data without sacrificing functionality. The bit sequences are amenable for use on embedded systems and have been applied to ray-casting and SLAM.

A. Future Work

VOLA is now been developed so that any information can be automatically added and unpacked from the two bit per voxel representation. This will allow data to be sent for specific applications for the same database. VOLA is also being merged with existing GIS data so that a curated training set can be developed for training CNNs on real-world objects.

References


