QFib: Fast and Efficient Brain Tractogram Compression
Corentin Mercier, S. Rousseau, P. Gori, I. Bloch, T. Boubekeur

To cite this version:

HAL Id: hal-02454772
https://hal.telecom-paris.fr/hal-02454772
Submitted on 24 Jan 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
QFib: Fast and Efficient Brain Tractogram Compression

C. Mercier · S. Rousseau · P. Gori · I. Bloch · T. Boubekeur

Received: date / Accepted: date

Abstract Diffusion MRI fiber tracking datasets can contain millions of 3D streamlines, and their representation can weight tens of gigabytes of memory. These sets of streamlines are called tractograms and are often used for clinical operations or research. Their size makes them difficult to store, visualize, process or exchange over the network. We propose a new compression algorithm well-suited for tractograms, by taking advantage of the way streamlines are obtained with usual tracking algorithms. Our approach is based on unit vector quantization methods combined with a spatial transformation which results in low compression and decompression times, as well as a high compression ratio. For instance, a 11.5GB tractogram can be compressed to a 1.02GB file and decompressed in 11.3 seconds. Moreover, our method allows for the compression and decompression of individual streamlines, reducing the need for a costly out-of-core algorithm with heavy datasets. Last, we open a way toward on-the-fly compression and decompression for handling larger datasets without needing a load of RAM (i.e. in-core handling), faster network exchanges and faster loading times for visualization or processing.

Keywords Compression · Diffusion MRI · Tractography · On-the-fly algorithms · Unit vectors

1 Introduction

Diffusion magnetic resonance imaging (dMRI) tractography is currently the only technique able to non-invasively ob-

tain the white matter architecture of the human brain. Tractography helps clinicians, neurosurgeons and researchers to understand the connections of the brain and is widely used for pre-operative planning, during clinical operations, and for research purposes. Fiber tracking datasets – called tractograms – are composed of 3D streamlines represented as 3D polylines with hundreds to thousands of ordered 3D points. Modern tractography algorithms can obtain up to several millions of these streamlines (Tournier et al. (2011)), resulting in tens of gigabytes (GB) of data (Rheault et al. (2017)). For instance, a file containing 1 million streamlines and obtained with a constant stepsize of 0.1mm can weight up to 8.7GB. This massive amount of data complicates visualization, processing, sharing or storage. In this article, we introduce a new compression algorithm for fiber tracking datasets, which is both fast and efficient.

Existing methods, that propose a solution to this data size problem, can be divided into two different categories. They either compress the whole tractogram or the representation of each streamline.

Tractogram-level compression This kind of approach consists in reducing the number of streamlines of a tractogram. A simple way of doing so is to randomly select a subset of streamlines from the original dataset (Gori et al. (2016)). This is a pragmatic approach, however, there is no guarantee that meaningful streamlines are not removed. Another method is based on grouping similar streamlines into clusters (Alexandroni et al. (2017); Garyfallidis et al. (2012); Guevara et al. (2011); Liu et al. (2012); Maddah et al. (2007); Wassermann et al. (2010); Demir and Çetingül (2015); Zhang et al. (2018); Siless et al. (2018)) in order to remove statistical redundancy. In that case, clusters can be represented using a streamline from the original dataset – usually called a prototype (Guevara et al. (2011)) – or with one created for this purpose (Garyfallidis et al. (2012)).
These clusters are built using metrics specifically designed for brain streamlines, thus preserving important properties such as extremities and path (Olivetti et al. (2017); Siless et al. (2018)). Another way of grouping streamlines is by doing it gradually, merging similar streamlines together, and thus obtaining a multiresolution representation (Mercier et al. (2018); Zhang and Laidlaw (2002)). A geometric representation of grouped streamlines can also be used (Madidah et al. (2007)), such as generalized cylinders (Mercier et al. (2018); Petrovic et al. (2007)) so that the spatial extent of the merged streamlines are kept at each resolution. Using these approaches, the data will be easier to understand and visualize, or even to analyze according to certain criteria (general shapes, connections, etc.). Nevertheless, depending on the application, keeping all the original streamlines might be required or preferred.

Streamline-level compression When trying to keep as much information as possible from the original tractogram, per-streamline compression algorithms appear to be a good alternative. These methods can, in most cases, be combined with a tractogram-level compression.

It is possible to use general compression algorithms directly on the data. Whereas lossless compression algorithms compress data by identifying and removing statistical redundancy — and therefore do not lose any information from the original data — lossy algorithms are going further by removing unnecessary data or introducing approximations. In the case of brain tractograms, the data are composed of a set of 3D points expressed as floating point values. A pragmatic approach is to compress them using a general floating point data compression algorithm such as the one of Lindstrom (2014). However, this approach cannot compete with more data-aware methods, which are more efficient as they take advantage of the specificities of the data. For instance, a classical approach in neuroscience is to reduce the number of streamlines. In Alexandroni et al. (2017) the idea is to use a dictionary combined with the fiber-density-coreset method. As a result, it removes high-frequency data (sudden changes in the streamlines path are smoothed) by only retaining a few non-zero coefficients for each streamline. The typical average error is around 2 mm on data with a resolution of 1.25 mm³. In Caiafa and Pestilli (2017), a sparse encoding is performed on the tensor representation. This results in a size in memory which is about the same as the one with usual 3D points representation. Moreover, points are moved at the center of the voxels of the original grid, resulting in possible important errors, and point order is lost in the process.

The maximum error when compressing streamlines should not exceed the voxel size, and should ideally be an order of magnitude smaller. This should be the case only with clinical DWI, whose voxel size are around 2 mm³, but also for research protocols (e.g. 1.25 mm³ in Van Essen et al. (2012)). Moreover, these dictionary-based methods do not allow for the compression and decompression of specific fascicles or tracts, which could drastically reduce the memory load for visualization or processing.

Consequently, there is an important need for a new efficient and fast compression algorithm for brain tractography data. To combine speed, compression ratio, the ability to compress independently each streamline of the tractogram, and the scalability, we focus our work on the representation of individual streamlines and introduce a new compression and decompression algorithm based on this representation.

Data constraints for our representation To compress data, it is necessary to find a space of representation smaller than the original one. In our case, we take advantage of the way streamlines are acquired. In particular, we use two constraints: a constant stepsize (δ) and a maximum angle of deviation (ψ).

Commonly used tractography algorithms use a constant stepsize while tracking streamlines. This property is some-
times used in some post-processing (Soares et al. (2013)). Furthermore, termination of streamlines happens when the angle between two consecutive segments of the streamline is higher than a maximum angle. Using these two constraints allows us to consider a streamline as a succession of unit vectors whose angles between consecutive vectors are bounded by the maximum angle.

Unit vectors quantization When considering lossy compression of independent unit vectors, the literature has shown that unit vectors quantizations are the most efficient (Cigolle et al. (2014)). They consider unit vectors as points on the surface of the unit sphere. A unit vector is compressed (or quantized) by being approximated with the closest point of a point set distributed onto the surface of the unit sphere, as shown in Figure 1. Each point is then encoded using an identifier that is smaller in memory than its original 3D coordinates. To minimize the maximal error (angle between the original unit vector and the quantized unit vector) the chosen point set needs to have a distribution as uniform as possible onto the surface of the unit sphere. Among possible distributions, octahedral quantization (Meyer et al. (2010)) is a good candidate for a fast compression and decompression of the unit vectors (Cigolle et al. (2014)). More recently, Keinert et al. (2015) introduced a new inverse mapping on the unit sphere, as shown that unit vectors quantizations are the most efficient (Cigolle et al. (2014)). They consider unit vectors as points on the surface of the unit sphere, and a unit vector quantization. This method has the following properties:

- High compression ratio (between 80 and 90%).
- Fast compression and decompression (few seconds).
- Low compression error.
- Ability to compress/decompress individual streamlines.
- Conservation of the number of points of each streamline.
- Ability to access any random single streamline from the compressed dataset.

Moreover, we provide a publicly available open source implementation of our compression and decompression algorithm along with a docker file.

2 Method

We define $\delta$ as the constant stepsize and $\psi$ as the maximum angle in the following parts of this article.

2.1 Brain streamlines representation

A streamline $f_a$ containing $N_a$ points is described as a 3D polyline $f_a = \{p_1 \ldots p_{N_a}\}$ of ordered points. Each streamline has its own number of points $N_a$ as we use a constant stepsize $\delta_a$. The $i^{th}$ point of the streamline is defined as $p_i = p_{i-1} + \delta_a \hat{p}_{i-1}$ for $i \in [2, N_a]$. Since two successive points along the streamline are supposed to be at a constant distance, $p_i$ can also be written as:

$$p_i = p_{i-1} + \delta_a \frac{\hat{p}_{i-1}}{|\hat{p}_{i-1}|}$$ (1)

where $\hat{p}$ denotes the normalized vector $\hat{p} = \|\hat{p}\|$. This means that each 3D polyline from the dataset can be represented by its first point and a set of unit vectors. The stepsize needs to be constant on a per-streamline basis for this representation to work.

2.2 Unit vectors quantization

Using unit vectors already reduces the size required to store streamlines, as only two dimensions (spherical coordinates)
Octahedral quantization (Meyer et al. 2010) It is a unit vector representation method that projects the vector $[x, y, z]$ defined on the surface of the unit sphere to an octahedron by normalizing it using an L1-norm. This octahedron is then unwrapped onto a 2D unit square as shown in Figure 2. The resulting 2D coordinates $[u, v]$ are discretized prior to encoding. When encoding using an $M$ bits quantization, each of these two coordinates is discretized on $M/2$ bits. The unquantized vector can be retrieved using the following formula (Meyer et al. 2010), where $\sigma(x) = 1$ for $x \geq 0$ and $\sigma(x) = -1$ otherwise:

$$z = 1 - |u| - |v|$$

$$[x, y] = \begin{cases} \quad [u, v]^T & \text{if } z \geq 0 \\ \quad [\sigma(v) - v, \sigma(u) - u]^T & \text{if } z < 0 \end{cases}$$ (2)

Spherical Fibonacci quantization (Keinert et al. 2015) This method is based on the Spherical Fibonacci point set as it yields a nearly uniform point distribution on the surface of the unit sphere (Keinert et al. 2015). Using this point set, the spherical coordinates $(\phi, \theta)$ of the $j^\text{th}$ point of the point set containing $K$ points are defined as (González 2010):

$$\begin{align} \theta_j &= \arccos\left(1 - \frac{2j+1}{K}\right) \\ \phi_j &= 2j\pi((3 - \sqrt{5})/2) \end{align}$$ (3)

When compressing a vector using an $M$ bits quantization, the number of points of discretization used will be $K = 2^M$. The mapping proposed by Keinert et al. 2015 is used to find the closest spherical Fibonacci point to the unit vector to encode (Figure 1) and the identifier $j$ of this closest point is stored. The advantages of both quantization methods are discussed in Section 3.

2.3 Mapping

We showed in Section 2.2 that quantized unit vectors can be used to represent streamlines. This representation can be considered as a differential description of the streamline for which the decompression is performed by an integration using the first point and the stepsize $\delta$ as constant terms. It provides good compression ratios, but it is possible to even push forward the compression by adding the constraint on the maximal angle $\psi$ between two consecutive segments. We take advantage of this property by expressing each unit vector with respect to the previous one. It can be seen as a second order differential representation (second derivative). The relative position between consecutive unit vectors is limited to a spherical cap, parametrized by the angle $\psi$ (in yellow in Figure 3). As such, the quantization point set would only be partially used, which is not optimal. Defining a uniform point set on a spherical cap could solve this issue, however, none exists (Rousseau and Boubekeur 2017).

Another possibility is to define a mapping from the spherical cap to the whole unit sphere. In this way, vectors could be quantized on the whole unit sphere. To be retrieved, vectors will then need to be unquantized before being unmapped. To minimize the error, this mapping should preserve the uniformity of the point set distribution. Such a mapping was recently introduced by Rousseau and Boubekeur (2017). As illustrated in Figure 3, the conservation of the point set distribution is ensured by keeping the equality between the ratio $dS_1/S_{\text{cap}}$ and $dS_2/S_{\text{sphere}}$ where $dS_1$ is an infinitesimal ring on the surface of the spherical cap (orange), $dS_2$ is the infinitesimal ring on the whole sphere corresponding to $dS_1$ mapped to the whole sphere.
(blue), and $S_{\text{cap}}$ and $S_{\text{sphere}}$ are respectively the surface of the spherical cap (yellow) and of the whole unit sphere (light blue). This method can be adapted to our problem by encoding each unit vector from its predecessor. A point $x$ on a spherical cap is transformed into a point $x'$ on the whole unit sphere by:

$$x' = cP_0 + \sqrt{1-c^2} \quad \text{with} \quad c = 1 - \frac{1 - \cos(\theta_1)}{k} \quad (4)$$

where $k = \frac{1 - \cos(\psi)}{2} + \varepsilon$ is the ratio between the length of the projection of the spherical cap on axis $P_0$ and the diameter of the unit sphere (Figure 3). In practice, the unit vector $\vec{P}_i \rightarrow \vec{P}'_i$ is mapped relative to $\vec{P}_i \rightarrow \vec{P}'_{i-1}$ to find the vector $x'$, by using $P_0' = \vec{P}_0 \rightarrow \vec{P}'_{i-1}$ and $x = \vec{P}_i \rightarrow \vec{P}'_{i-1}$. As indicated in the original article by Rousseau and Boubekeur (2017), we add an $\varepsilon$ to the ratio due to numerical imprecision to ensure that all unit vectors lie on the spherical cap. Representing unit vectors relative to the previous ones can be seen as the second derivative, as the first point is needed for the first integration and the first unit vector for the second one.

2.4 Propagation error reduction

Each time a unit vector is compressed, a small error is introduced due to the quantization process, meaning that starting from the point $p_{i-1}$, the new position will be $p'_i$ instead of $p_i$. Therefore, continuously compressing the unit vectors $p_{i-1} \rightarrow p'_i$ can lead to an important error as shown with the orange curve in Figure 4. To avoid this error from being fully propagated, we substitute the points $p_{i-1}$ for $p'_i$. This point has already been mapped and quantized (compressed) at the previous step, and then unquantized and unmapped (decompressed). As a result, it is not at the exact same place as $p_{i-1}$, and instead of compressing the unit vectors $p_{i-1} \rightarrow p'_i$, we compress the unit vectors $p'_i \rightarrow p'_i$. This slight modification compensates for the accumulated error. Indeed, it allows the streamline being compressed to always use a direction related to its current position, thus reducing the error propagation. This is presented on the green curve of Figure 4. In Section 2, we obtain the following algorithm:

- we store the first and second points as they are, using their Cartesian coordinates;
- for the other points, we use the mapping proposed by Rousseau and Boubekeur (2017) on the unit vector $\vec{P}_{i-1} \rightarrow \vec{P}'_{i-1}$, where the axis of the spherical cap is $\vec{P}_{i-2} \rightarrow \vec{P}'_{i-2}$, and the ratio depends on the maximum angle $\psi$;
- we quantize the mapped vector using the octahedral (Meyer et al. (2010)) or the spherical Fibonacci quantization (Keinert et al. (2015)) depending on the application scenario.

This algorithm is applied independently on each streamline of the dataset. As the first unit vector is recomputed for each streamline, it allows for a different $\delta_i$ for each streamline, making qfib usable for resampled streamlines, as long as it guarantees constant stepsize along each streamline. The C++ pseudo code for the compression and decompression is available in Appendix A. When used for file storage, we define the qfib format in Figure 6. We evaluate it in Section 3.

3 Results and discussion

We test our algorithm using streamlines traced from a randomly selected subject of the Human Connectome Project (Van Essen et al., 2012) dataset. As our method requires a constant stepsize, we use the SD_STREAM (deterministic)
and iFOD1 (probabilistic) algorithms from MRtrix (Tournier et al. (2012)) to compute the tractograms. We test the effectiveness of our algorithm on 12 bundles where we vary the stepsize \( \delta \) between 0.1 and 1 mm, and the number of streamlines between 60k and 3M. The maximum angle is not specified when generating the streamlines, which means that the default formula of MRtrix (Tournier et al. (2012)) is used: \( \text{maximum angle} = 90^\circ \times \delta / \text{voxelsize} \). The voxels size on the dMRI images is 1.25mm\(^3\). Streamlines lengths are constrained between 40 mm and 256 mm and are forced to end on the brain cortical surface. All configurations are presented in Table 1 with file sizes (tck format).

Our technique is general and any diffusion model or tractography algorithm could be used provided it is based on a per-streamline constant stepsize \( \delta \). We implemented a TCK file reader but any file format for 3D points can be handled, assuming a suitable reader is provided to access to a list of point coordinates. Although its application is demonstrated with the octahedral and spherical Fibonacci quantizations, our method is compatible with any actual or future unit vector quantization method.

We first illustrate the results of our method on a few toy examples, then compare our algorithm with zfib (Presseau et al. (2015)). We evaluate the compression ratios of the two approaches and their impact on mapped scalar values, using fractional anisotropy (FA) as an example. We conclude by showing that our method can also run out-of-core.

### 3.1 Error

To better illustrate the error introduced using our compression method, we show in Figure 7 four different streamlines (in black) and their compressed and decompressed version (in green). As we quantize the relative directions of each segment, a straight streamline (Figure 7(a)) remains totally straight, with no error introduced. However, when introducing some curvature in the streamlines, we notice that the error increases according to the angle as shown in Figures 7(b), 7(c), 7(d), for which the maximum angles are respectively 40.3°, 45.7° and 73.7°. With the spiral curve (Figure 7(d)), the angle is too important for the mapping to be effective. In that case, the error is higher, and the difference with the original curve is more visible than with the other curves. However, this streamline is not anatomically meaningful as such curvature should not appear in a tractogram.

Our compression keeps the original number of points. As a result, we are able to measure pointwise errors. The errors are computed between the original dataset containing the streamlines \( f_a = \{p_{a,1}, \ldots, p_{a,N_a}\} \) and the compressed and decompressed streamlines \( f'_a = \{p'_{a,1}, \ldots, p'_{a,N_a}\} \) using the following formulas:

\[
\text{mean error} = \frac{\sum_{a=1}^{L} \sum_{b=1}^{N_a} ||p_{a,b} - p'_{a,b}||}{\sum_{a=1}^{L} N_a}
\]

\[
\text{max error} = \max \{||p_{a,b} - p'_{a,b}|| \mid a \in [1,L], b \in [1,N_a]\}
\]

where \( N_a \) is the number of points in the streamline \( f_a \), and \( L \) the number of streamlines in the dataset. Table 3 shows these errors for every file described in Table 1. It emphasizes that increasing the precision of the quantization from 8 bits to 16 bits greatly reduces the error as there are more possible points on the unit sphere during quantization. However, this results in lower compression ratios (see Section 3.2).

Even though it is usually advised to use stepsizes at a tenth of the voxel size (so around 0.1 or 0.2 mm in our case), greater stepsizes can be used. In these cases, according to our errors, it is better to use a 16 bits quantization. The method used for quantization impacts the error. The spherical Fibonacci quantization introduces less error, but octahedral quantization remains interesting for its faster computation time with still relatively low error (see Section 3.3).

Figure 8 evaluates the error depending on streamlines length on a histogram where each column represents a cluster of 10 mm length difference. The tractograms were made using a probabilistic algorithm and a 0.1 mm stepsize. They were compressed and decompressed using an 8 bits octahedral quantization. The histogram shows that the maximum error seems to be mostly independent of the length, in contrast to the average error which increases with the length of the streamlines.

Table 2 shows the maximum and average errors obtained at the extremities of the streamlines. This point is important as it characterizes the connectivity of the streamline. We notice that the error (including the maximum one) is really low for \( \delta = 0.2 \) mm with an 8 bits quantization. For \( \delta = 0.5 \) mm, the error starts to be too important, especially with the octahedral quantization, and a 16 bits quantization is preferable.

---

**Fig. 6: qfib** – The format used to store the compressed streamlines.

**Table 1: Size of the files (tck format).**

<table>
<thead>
<tr>
<th>Stepsize ( \delta )</th>
<th>0.1 mm</th>
<th>0.2 mm</th>
<th>0.5 mm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nb. of streamlines</td>
<td>500k</td>
<td>3M</td>
<td>500k</td>
</tr>
<tr>
<td></td>
<td>3M</td>
<td>3M</td>
<td>3M</td>
</tr>
<tr>
<td><strong>Deterministic</strong></td>
<td>1.02</td>
<td>1.02</td>
<td>0.44</td>
</tr>
<tr>
<td><strong>Probabilistic</strong></td>
<td>0.86</td>
<td>0.91</td>
<td>0.51</td>
</tr>
</tbody>
</table>

*Size of input file (GB)*

*Compressed Fiber*

Header
- int8 for each point (8 bits)
- int16 for each point (16 bits)

Compressed File (.qfib)
- int8 for each point (8 bits)
- int16 for each point (16 bits)

<table>
<thead>
<tr>
<th>Nb. of streamlines</th>
<th>500k</th>
<th>3M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ratios float32</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>Quantizations uint8</td>
<td>0.5</td>
<td>0.5</td>
</tr>
</tbody>
</table>

*Compressed points*
- int8 for each point (8 bits)
- int16 for each point (16 bits)
To visualize these errors, we color-mapped the compressed streamlines according to their distance from the original dataset (Figure 9). The quantization used is the octahedral quantization. Indeed, the error obtained in this case (Figure 8) is almost not visible on the picture, compared to the 8 bits precision (Figure 9(a)).

Table 2: Maximum and average errors on the endpoints.

<table>
<thead>
<tr>
<th>Stepsize δ (mm)</th>
<th>0.1 mm</th>
<th>0.2 mm</th>
<th>0.5 mm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>500k</td>
<td>3M</td>
<td>500k</td>
</tr>
<tr>
<td>Quantization</td>
<td>Precision</td>
<td>Deterministic</td>
<td>Precision</td>
</tr>
<tr>
<td>Fibonacci</td>
<td>8 bits</td>
<td>4.94</td>
<td>5.30</td>
</tr>
<tr>
<td></td>
<td>16 bits</td>
<td>0.50</td>
<td>0.51</td>
</tr>
<tr>
<td>Octahedral</td>
<td>8 bits</td>
<td>7.53</td>
<td>8.03</td>
</tr>
<tr>
<td></td>
<td>16 bits</td>
<td>0.30</td>
<td>0.36</td>
</tr>
<tr>
<td>Quantumization</td>
<td>8 bits</td>
<td>3.04</td>
<td>2.95</td>
</tr>
<tr>
<td></td>
<td>16 bits</td>
<td>0.13</td>
<td>0.14</td>
</tr>
<tr>
<td>Fibonacci</td>
<td>8 bits</td>
<td>4.79</td>
<td>4.90</td>
</tr>
<tr>
<td></td>
<td>16 bits</td>
<td>0.13</td>
<td>0.14</td>
</tr>
<tr>
<td>Octahedral</td>
<td>8 bits</td>
<td>2.34</td>
<td>2.56</td>
</tr>
<tr>
<td></td>
<td>16 bits</td>
<td>0.10</td>
<td>0.09</td>
</tr>
<tr>
<td>Quantumization</td>
<td>8 bits</td>
<td>1.35</td>
<td>0.22</td>
</tr>
<tr>
<td></td>
<td>16 bits</td>
<td>0.04</td>
<td>0.01</td>
</tr>
<tr>
<td>Fibonacci</td>
<td>8 bits</td>
<td>2.69</td>
<td>0.45</td>
</tr>
<tr>
<td></td>
<td>16 bits</td>
<td>0.04</td>
<td>0.01</td>
</tr>
</tbody>
</table>

Table 3: Maximum and average errors of our method depending on the dataset, precision in bits and quantization method.

<table>
<thead>
<tr>
<th>Stepsize δ (mm)</th>
<th>0.1 mm</th>
<th>0.2 mm</th>
<th>0.5 mm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>500k</td>
<td>3M</td>
<td>500k</td>
</tr>
<tr>
<td>Quantization</td>
<td>Precision</td>
<td>Deterministic</td>
<td>Precision</td>
</tr>
<tr>
<td>Fibonacci</td>
<td>8 bits</td>
<td>4.94</td>
<td>5.30</td>
</tr>
<tr>
<td></td>
<td>16 bits</td>
<td>0.50</td>
<td>0.51</td>
</tr>
<tr>
<td>Octahedral</td>
<td>8 bits</td>
<td>7.53</td>
<td>8.03</td>
</tr>
<tr>
<td></td>
<td>16 bits</td>
<td>0.30</td>
<td>0.36</td>
</tr>
<tr>
<td>Quantumization</td>
<td>8 bits</td>
<td>3.04</td>
<td>2.95</td>
</tr>
<tr>
<td></td>
<td>16 bits</td>
<td>0.13</td>
<td>0.14</td>
</tr>
<tr>
<td>Fibonacci</td>
<td>8 bits</td>
<td>4.79</td>
<td>4.90</td>
</tr>
<tr>
<td></td>
<td>16 bits</td>
<td>0.13</td>
<td>0.14</td>
</tr>
<tr>
<td>Octahedral</td>
<td>8 bits</td>
<td>2.34</td>
<td>2.56</td>
</tr>
<tr>
<td></td>
<td>16 bits</td>
<td>0.10</td>
<td>0.09</td>
</tr>
<tr>
<td>Quantumization</td>
<td>8 bits</td>
<td>1.35</td>
<td>0.22</td>
</tr>
<tr>
<td></td>
<td>16 bits</td>
<td>0.04</td>
<td>0.01</td>
</tr>
<tr>
<td>Fibonacci</td>
<td>8 bits</td>
<td>2.69</td>
<td>0.45</td>
</tr>
<tr>
<td></td>
<td>16 bits</td>
<td>0.04</td>
<td>0.01</td>
</tr>
</tbody>
</table>

3.2 Compression Ratio

The compression ratio is computed using the following formula:

\[
\text{compression ratio} = 100 \times \left(1 - \frac{\text{compressed size}}{\text{original size}}\right)
\]

Our compression ratio (Table 4) is mostly stable around 90% using an 8 bits quantization and around 82% using a 16 bits quantization. We emphasize that our compression ratios are strictly better than zfib for the same error values with an 8 bits quantization and stepsizes under 0.5mm which correspond to the advised values for the stepsize. When the zfib error is set to 0.2mm (their default value), we have either better compression ratios or smaller errors. For the streamlines generated using a probabilistic method, that are
more tangled, we achieve better compression ratios with the same error value in every shown case.

3.3 Computation time

In Table 5, we show the compression and decompression times of zfib and qfib. These figures were obtained using a computer with an Intel Xeon E5-1650v4 (6 cores, 12 threads, 3.6GHz). In both methods, we only account for the compression and decompression time and not reading and writing from/to the hard drive. We set the maximal error to 0.2 mm. N/A are the values for which the algorithm was not able to perform the full compression and decompression.

3.4 Impact on Fractional Anisotropy

Scalar quantities such as Fractional Anisotropy (FA) are often mapped onto the streamlines. As such, it is necessary to verify that their values remain respected even when streamlines are compressed. We compute the FA map using a Bresenham-like integration similarly to Presseau et al. (2015). Results are presented in Table 6 as an error percentage between the original streamlines and the compressed and decompressed ones. The error used for Presseau’s algorithm was the default one of 0.2 mm. This table shows that we obtain a lower average FA error in all tested configurations.

To go even further, we compute the FA error pointwise similarly to Equation 5. We use a trilinear interpolation of the 8 voxels surrounding each point. This cannot be done for zfib as it does not keep the same amount of points as the original dataset. The result of this comparison is presented in Table 7. We can see that our maximum errors are under 0.2mm for 8 bits quantization when using stepsizes less than 0.5mm. Moreover, the average error is really low, which was expected considering the low errors reported in Table 6.
Table 6: Comparison of zfib and qfib for FA computation.

We compute the average error in FA computation using a Bresenham-like integration (Presseau et al. (2015)).

<table>
<thead>
<tr>
<th>Stepsize $\delta$</th>
<th>No. of streamlines</th>
<th>0.1 mm 500k</th>
<th>0.2 mm 500k</th>
<th>0.5 mm 500k</th>
<th>0.1 mm 3M</th>
<th>0.2 mm 3M</th>
<th>0.5 mm 3M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method</td>
<td>Bits</td>
<td>Deterministic</td>
<td>Probabilistic</td>
<td>Probabilistic</td>
<td>Deterministic</td>
<td>Probabilistic</td>
<td>Probabilistic</td>
</tr>
<tr>
<td>zfib fibonacci</td>
<td>5</td>
<td>0.004 0.004 0.008 0.009 0.023 0.023</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>zfib octahedral</td>
<td>8</td>
<td>0.006 0.006 0.012 0.012 0.031 0.036</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>zfib</td>
<td>-</td>
<td>2.506 2.527 2.572 2.553 2.290 2.219</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7: Errors of FA computation between original streamlines and the compressed and decompressed ones using qfib.

<table>
<thead>
<tr>
<th>Stepsize $\delta$</th>
<th>No. of streamlines</th>
<th>0.1 mm 500k</th>
<th>0.2 mm 500k</th>
<th>0.5 mm 500k</th>
<th>0.1 mm 3M</th>
<th>0.2 mm 3M</th>
<th>0.5 mm 3M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quantization</td>
<td>Precision</td>
<td>Deterministic</td>
<td>Probabilistic</td>
<td>Probabilistic</td>
<td>Deterministic</td>
<td>Probabilistic</td>
<td>Probabilistic</td>
</tr>
<tr>
<td>Fibonacci</td>
<td>8 bits</td>
<td>3.91 3.93 7.92 8.32 22.6 22.9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Octahedral</td>
<td>8 bits</td>
<td>5.89 6.23 12.9 12.4 32.7 32.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average error of FA computation (in percentage)</td>
<td></td>
<td>∀δ∈(0,1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.6 Limitations

The main limitation of our method is the requirement for a constant stepsize per streamline. To overcome this issue, one could previously use a resampling method. Furthermore, we conceived our method especially for brain tractography, using specific properties and constraints, thus other limitations could appear when trying to apply the same compression algorithm to other kinds of data. For instance, in the case of streamlines with a bigger maximum angle between consecutive segments, the mapping might not be as efficient, as illustrated in Figure 7. In such a case, it could be necessary to use a 16 bit or higher precision quantization, thus reducing the resulting compression ratio.

3.7 Future work

Our method is pleasingly parallel, which means that it can be trivially parallelized at the streamline level. Therefore, it would be straightforward to implement a massively parallel version on the GPU. This could improve a lot the compression and decompression times. Doing so, an interesting subject of research will be to try to visualize the compressed streamlines, by individually decompressing them on-the-fly from the VRAM. This could reduce the need for an important amount of RAM and VRAM on the machine, and improve the data transfer speed between RAM and VRAM. In the future, it could be interesting to integrate the proposed method in existing tractography algorithms to generate streamlines directly in this compressed format. To fur-
ther improve the streamline compression, a more parsimonious space of representation could be used, using, for instance, Bézier curves, B-splines or NURBS.

4 Conclusion

We have presented a novel streamline compression algorithm for brain tractograms, and its associated encoding format – qfib. We evaluated and validated it using a wide variety of brain tractography configurations with different step-sizes and numbers of streamlines. This algorithm provides lower errors in general cases and better compression and decompression times than existing methods. The compression ratio is high, around 90%, with errors under the dMRI precision. In contrast to other methods, qfib does not remove any point from the original dataset, which is important when point-based algorithms need to be applied afterward. Moreover, the compression and decompression steps handle each streamline individually. This implies that they can both be easily parallelized and that qfib is as efficient on small as on large databases in contrast to dictionary-based methods. This also means that one can decompress single streamlines from the compressed representation, allowing users to directly access specific fascicles or tracts with negligible memory usage and loading time. Consequently, our approach opens a way toward on-the-fly application where an algorithm could work with the compressed representation of the streamlines in memory, and decompress only single streamlines or small bundles before applying some computation on them. This reduces the need for loads of RAM or VRAM. We also demonstrated that our algorithm works out-of-core. In this case, the size of the data to compress is only limited by the hard drive.

We provide a reference C++ implementation of our method as an open source code on GitHub2.

Acknowledgements We would like to thank Alessandro Delmonte for his precious help and his feedback on our results. This work was partially supported by the French National Research Agency (ANR) under grant ANR 16-LCV2-0009-01 ALLEGRO, by the DGA and by Labex DigiCosme (project ANR11LABEX0045DIGICOSME) operated by ANR as part of the program Investissement d’Avenir Idex ParisSaclay (ANR11IDEX000302).

Conflict of interest

The authors declare that they have no conflict of interest.

References


---

2 https://github.com/syrousseau/qfib
A Pseudo-code

```cpp
struct CFiber {
    vec3 first, second;
    vector<uint8_t> data;
}

void compress(const vector<vec3>& fiber, CFiber &cfiber) {
    // encode the first two points
    cfiber.first = fiber[0];
    cfiber.second = fiber[1];

    // compute the first unit vector and the stepsize
    vec3 axis = normalize(fiber[1] - fiber[0]);
    float stepsize = distance(fiber[0], fiber[1]);
    vec3 current = fiber[1];

    // compress the rest of the fiber
    for (unsigned i = 2; i < fiber.size(); ++i) {
        vec3 v = normalize(fiber[i] - current);
        cfiber.data[i-2] = quantize(inverseMapping(v, axis, ratio))
        axis = mapping(unquantize(cfiber.data[i-2]), axis, ratio);
        current = current + (stepsize * axis);
    }
}

void decompress(const CFiber &cfiber, vector<vec3>& fiber) {
    // decode the first two points
    fiber[0] = cfiber.first;
    fiber[1] = cfiber.second;

    // compute first unit vector and stepsize
    vec3 axis = normalize(fiber[1] - fiber[0]);
    float stepsize = distance(fiber[0], fiber[1])

    // decode the rest of the fiber
    for (unsigned i = 0; i < cfiber.data.size(); ++i) {
        vec3 v = mapping(unquantize(cfiber.data[i]));
        fiber[i+2] = fiber[i+1] + (stepsize * v);
        axis = v;
    }
}
```

C++ pseudocode of our compression and decompression algorithm. The pseudocode of the `mapping` and `inverseMapping` functions can be found in [Rousseau et al. 2017] article. The `quantize` and `unquantize` functions are the octahedral quantization [Meyer et al. 2010] or the spherical Fibonacci [Keinert et al. 2015]. For more details on the implementation, the reader can refer to the source code available on GitHub [https://github.com/syrousseau/qfib](https://github.com/syrousseau/qfib)