QFib: Fast and Efficient Brain Tractogram Compression

Corentin Mercier, S. Rousseau, P. Gori, Isabelle Bloch, T. Boubekeur

Preprint manuscript No.
(will be inserted by the editor)

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 obtain 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 ¸Cetingül (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)).

⋆ C. Mercier · S. Rousseau contributed equally to this work.

C. Mercier · S. Rousseau · P. Gori · I. Bloch · T. Boubekeur
LTCI, Télécom Paris, Institut Polytechnique de Paris

C. Mercier
LIX, École Polytechnique, Institut Polytechnique de Paris
Corresponding authors : {sylvain.rousseau|corentin.mercier}@telecom-paris.fr – +33145817539

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 (Maddah 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 points, by using algorithms such as a linearization that removes colinear (or almost colinear) points (Presseau et al. (2015)). This can be combined with a general compression algorithm as data are still floating point values, improving the compression ratio.

One possible approach for a data-aware compression is to build a dictionary in a dedicated space of representation (Presseau et al. (2015); Kumar and Desrosiers (2016); Moreno et al. (2017)). Presseau's algorithm (Presseau et al. (2015)) uses a three steps process, starting with (i) linearization, then (ii) quantization and eventually (iii) encoding the points using a dictionary. The method is similar to the `zip` compression, and the algorithm is named `zfib`. This method achieves good compression ratios (around 90%), but at the cost of high compression times (more than ten minutes for tractograms with 1M streamlines). Some points from the original streamlines are removed due to the linearization step, therefore reducing the performance of point-based algorithms (Soares et al. (2013)). The authors also tried to apply some transformations – including wavelet-based ones – without success as they reduced the compression ratio.

Another compression algorithm method dedicated to tractography uses a sparse representation of streamlines (Chung et al. (2009); Kumar and Desrosiers (2016); Moreno et al. (2017)). In Chung et al. (2009), the streamlines are represented using cosine functions, parametrized with 60 parameters, reducing the memory cost. The average error introduced in this case is about $0.26mm$. Kumar and Desrosiers (2016)'s approach segments and clusters streamlines using a dictionary built with a sparse coding of the streamlines. In that case, the goal is not to reduce the memory usage but to ease some heavy computations on 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 $2mm$ on data with a resolution of $1.25mm^3$. 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 not only with clinical DWI, whose voxel size are around $2mm^3$, but also for research protocols (e.g. $1.25mm^3$ 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-

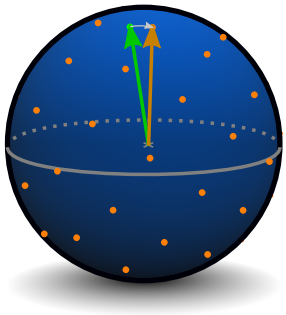


Fig. 1: A unit vector (in green) is approximated by the closest quantization point (in orange).

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 for the spherical Fibonacci point set. This point set is known for the uniformity of its distribution over the surface of the unit sphere. This quantization will, therefore, decrease the quantization error, but at the cost of computational complexity.

Both distributions make sense with our approach as we can either want a fast decompression for streaming algorithms (octahedral) or sacrifice a little bit of speed (spherical Fibonacci) to decrease the compression error that is directly linked to the compression ratio. These two methods will be used in the remainder of this article. The common point of all unit vector quantization techniques is their speed. These algorithms can usually decompress millions of unit vectors per second on a single processor (Cigolle et al. (2014)) and billions of them on a graphics processor unit (GPU).

Contribution In this article, we introduce a new representation model for the streamlines of brain tractograms. We then propose `qfib`, a new compression and decompression algorithm based on the proposed representation. It handles each streamline individually using two constraints specific to tractography methods (constant stepsize and maximum angle between consecutive points), a mapping on 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[1].

2 Method

We define δ as the constant stepsize and ψ 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 $f_a = \{p_1 \dots p_{N_a}\}$ of ordered points. Each streamline has its own number of points N_a as we use a constant stepsize δ_a. The i^{th} point of the streamline is defined as $p_i = p_{i-1} + \overrightarrow{p_{i-1}p_i}$ 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:

$$p_i = p_{i-1} + \delta_a \, \hat{\overrightarrow{p_{i-1}p_i}} \tag{1}$$

where $\hat{\overrightarrow{x}}$ denotes the normalized vector $\overrightarrow{x}/|\overrightarrow{x}|$. 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)

[1] `https://github.com/syrousseau/qfib`

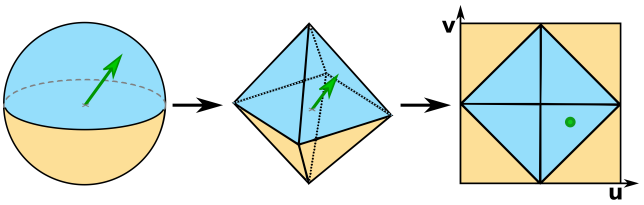


Fig. 2: Octahedral quantization (Meyer et al. (2010)) projects a unit vector to an octahedron, then, to a unit square to encode the discretization of the resulting 2D coordinates.

are needed to encode each segment instead of the three required by the Cartesian coordinates. When encoding the first two points of each streamline, their size, and unit vectors, the compression ratio will be slightly under 33%, depending on the average number of points in the streamline of the dataset. In order to further increase the compression ratio, we take advantage of the tractography resolution – linked to the dMRI resolution – and use a lossy compression. As introduced in Section 1, when choosing a quantization method, we can prioritize the speed with the *octahedral quantization* (Meyer et al. (2010)), or the precision with the *spherical Fibonacci quantization* (Keinert et al. (2015)).

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:

$$z = 1 - |u| - |v|$$
$$[x,y] = \begin{cases} [u,v]^T & \text{if } z \geq 0 \\ [\sigma(v) - v, \sigma(u) - u]^T & \text{if } z < 0 \end{cases} \quad (2)$$

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 (ϕ,θ) of the j^{th} point of the point set containing K points are defined as (González (2010)):

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

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

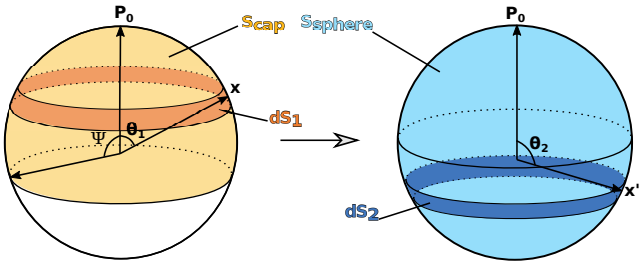


Fig. 3: Notations used in Section 2.3. The mapping introduced by Rousseau and Boubekeur (2017) transforms a point x defined on a spherical cap centered in P_0 and with a maximal angle of ψ represented as the yellow area on the surface of the unit sphere to another point x' defined on the surface of the whole unit sphere colored in light blue.

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 δ 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 ψ 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 ψ (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_{cap} and dS_2/S_{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

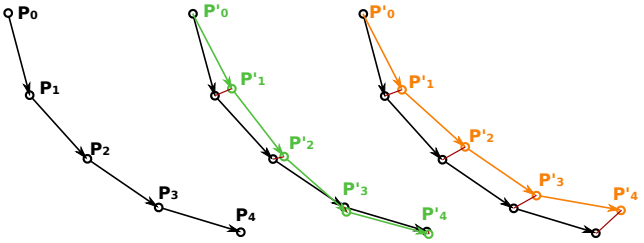


Fig. 4: Quantization with (in green) and without (in orange) propagation error reduction. The error computed per point is shown in red.

(blue), and S_{cap} and S_{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:

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

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 $\overrightarrow{\hat{P_{i-1}P_i}}$ is mapped relative to $\overrightarrow{\hat{P_{i-2}P_{i-1}}}$ to find the vector x', by using $P_0 = \overrightarrow{\hat{P_{i-2}P_{i-1}}}$ and $x = \overrightarrow{\hat{P_{i-1}P_i}}$. As indicated in the original article by Rousseau and Boubekeur (2017), we add an ε 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 $\overrightarrow{\hat{p_{i-1}p_i}}$ 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-1}. 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 $\overrightarrow{\hat{p_{i-1}p_i}}$, we compress the unit vectors $\overrightarrow{\hat{p'_{i-1}p_i}}$. 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

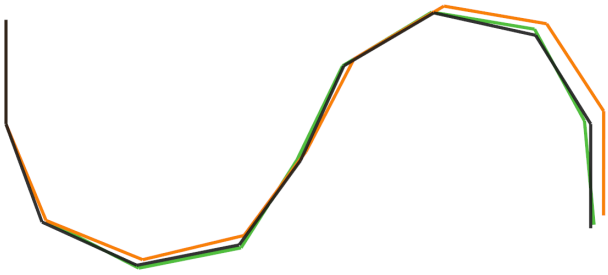


Fig. 5: Example on an S curve streamline (black). The compressed and decompressed streamline is shown without (in orange) and with error propagation reduction (in green).

Figure 5, we show on a small streamline example the difference between a streamline compressed without error propagation reduction (in orange) and with our error propagation reduction strategy (in green). The second one is closer to the original streamline (in black).

2.5 Compression algorithm and encoding pattern

To compress the streamline $f_a = \{p_1, ..., p_{N_a}\}$ into a streamline f' for which the slightly changed decompressed coordinates will be $\{p'_1, ..., p'_{N_a}\}$, combining all the steps described 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 $\overrightarrow{\hat{p'_{i-1}p_i}}$ where the axis of the spherical cap is $\overrightarrow{\hat{p'_{i-2}p'_{i-1}}}$, and the ratio depends on the maximum angle ψ;
- 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 δ_a 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)

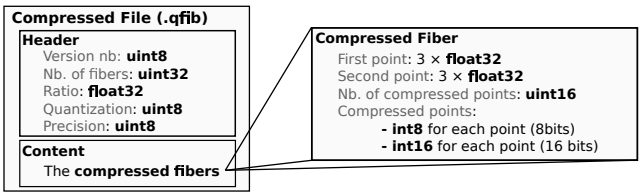


Fig. 6: `qfib` – The format used to store the compressed streamlines.

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

Stepsize δ	0.1 mm		0.2 mm		0.5 mm	
Nb. of streamlines	500k	3M	500k	3M	500k	3M
Size of input file (GB)						
Deterministic	3.80	22.8	1.92	11.5	0.84	5.01
Probabilistic	4.36	26.2	2.35	14.1	1.26	7.53

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 δ between 0.1 and $1mm$, 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: $maximum\ angle = 90° \times \delta / voxelsize$. The voxels size on the dMRI images is $1.25mm^3$. Streamlines lengths are constrained between $40mm$ and $256mm$ 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 (δ_a). We implemented a TCK file reader but any file fomat 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:

$$mean\ error = \frac{\sum_{a=1}^{L} \sum_{b=1}^{N_a} ||p_{a,b} - p'_{a,b}||}{\sum_{a=1}^{L} N_a} \qquad (5)$$
$$max\ error = max\{||p_{a,b} - p'_{a,b}|| \mid a \in [1,L],\ b \in [1,N_a]\}$$

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 $0.2mm$ 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 $10mm$ length difference. The tractograms were made using a probabilistic algorithm and a $0.1mm$ 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 $\delta_a \leq 0.2mm$ with an 8 bits quantization. For $\delta = 0.5mm$, the error starts to be too important, especially with the octahedral quantization, and a 16 bits quantization is preferable.

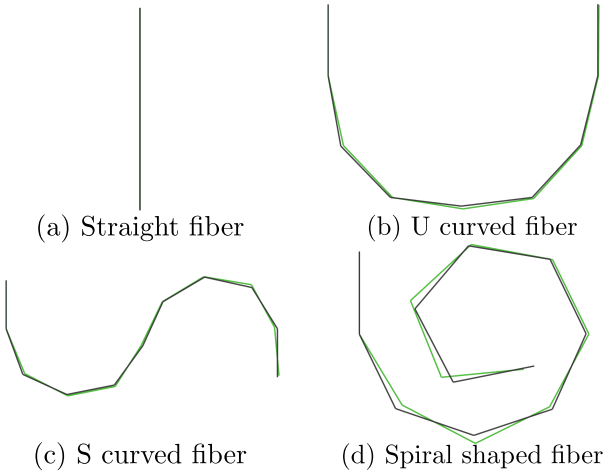


Fig. 7: Original streamlines (black), compressed and decompressed streamlines (green) using an 8 bits octahedral quantization.

Table 2: Maximum and average errors on the endpoints.

Stepsize δ		0.1 mm		0.2 mm		0.5 mm	
Nb. of streamlines		500k	3M	500k	3M	500k	3M
Maximum error ($\times 10^{-2} mm$)							
Quantization	Precision	Deterministic					
Fibonacci	8 bits	4.91	5.28	10.3	10.6	32.9	33.5
	16 bits	0.50	0.57	0.28	0.28	0.36	0.38
Octahedral	8 bits	7.53	8.03	15.9	16.5	46.1	50.7
	16 bits	0.50	0.56	0.26	0.29	0.47	0.46
Quantization	Precision	Probabilistic					
Fibonacci	8 bits	2.76	2.85	5.25	5.43	18.4	18.6
	16 bits	0.13	0.13	0.12	0.14	0.64	0.63
Octahedral	8 bits	4.78	4.78	8.30	8.64	26.7	31.8
	16 bits	0.13	0.13	0.14	0.15	0.79	0.80
Average error ($\times 10^{-3} mm$)							
Quantization	Precision	Deterministic					
Fibonacci	8 bits	19.9	19.9	42.2	42.2	109	109
	16 bits	0.44	0.44	0.32	0.32	1.17	1.18
Octahedral	8 bits	29.0	29.0	61.4	61.4	161	161
	16 bits	0.44	0.44	0.34	0.34	1.31	1.31
Quantization	Precision	Probabilistic					
Fibonacci	8 bits	17.5	17.5	25.1	25.3	62.5	62.7
	16 bits	0.21	0.21	0.41	0.41	2.12	2.12
Octahedral	8 bits	28.0	28.1	40.7	40.5	94.4	94.7
	16 bits	0.22	0.22	0.46	0.46	2.36	2.39

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 one. The color scale is going from $0mm$ (dark blue) to $1.25mm$ (dark red) of error, the voxel size of our data. We used the streamlines computed with a $1mm$ stepsize using the deterministic method. This case is an illustration of the situations where it is important to use a 16 bits precision quantization. Indeed, the error obtained in this case (Figure 9(b)) is almost not visible on the picture, compared to the 8 bits precision (Figure 9(a)).

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

Stepsize δ		0.1 mm		0.2 mm		0.5 mm	
Nb. of streamlines		500k	3M	500k	3M	500k	3M
Maximum error ($\times 10^{-2} mm$)							
Quantization	Precision	Deterministic					
Fibonacci	8 bits	4.94	5.30	10.3	10.6	33.2	34.1
	16 bits	0.50	0.57	0.28	0.28	0.39	0.38
Octahedral	8 bits	7.53	8.03	16.5	16.5	46.7	51.0
	16 bits	0.50	0.56	0.27	0.29	0.50	0.52
Quantization	Precision	Probabilistic					
Fibonacci	8 bits	3.04	2.95	5.86	5.91	19.7	20.6
	16 bits	0.13	0.14	0.14	0.16	0.69	0.72
Octahedral	8 bits	4.79	4.90	8.55	9.38	29.8	31.7
	16 bits	0.13	0.14	0.17	0.19	0.87	0.93
Average error ($\times 10^{-3} mm$)							
Quantization	Precision	Deterministic					
Fibonacci	8 bits	2.54	0.52	9.10	2.04	55.8	9.76
	16 bits	0.10	0.03	0.10	0.03	0.47	0.11
Octahedral	8 bits	3.12	0.55	12.4	2.29	66.8	18.8
	16 bits	0.10	0.03	0.10	0.03	0.64	0.15
Quantization	Precision	Probabilistic					
Fibonacci	8 bits	1.35	0.22	5.01	0.83	19.2	3.73
	16 bits	0.04	0.01	0.08	0.01	0.63	0.16
Octahedral	8 bits	2.69	0.45	5.21	1.11	37.5	6.26
	16 bits	0.04	0.01	0.08	0.02	0.94	0.20

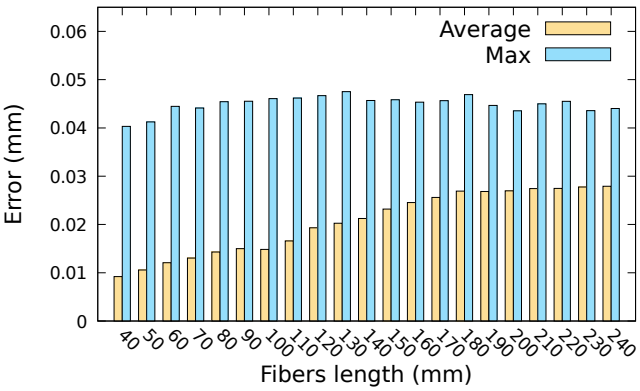


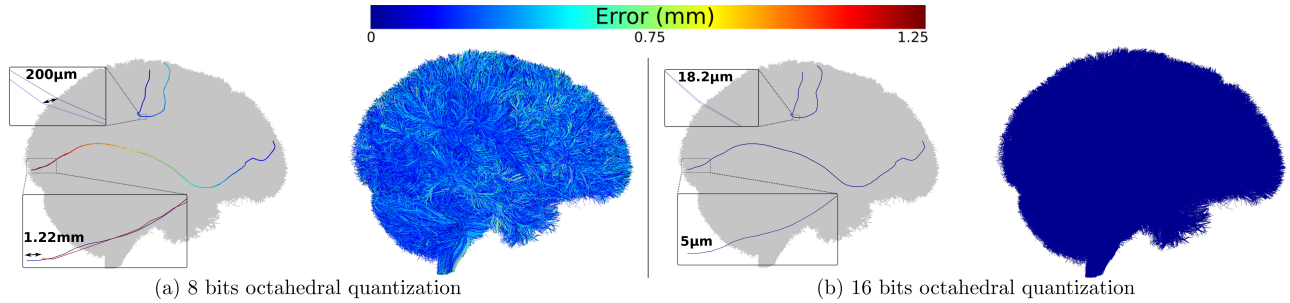
Fig. 8: Error in mm depending on streamlines length for an 8 bits octahedral quantization with $\delta = 0.1mm$.

3.2 Compression Ratio

The compression ratio is computed using the following formula:

$$compression\ ratio = 100 \times (1 - \frac{compressed\ size}{original\ size}) \quad (6)$$

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



(a) 8 bits octahedral quantization (b) 16 bits octahedral quantization

Fig. 9: The figure on the left of each panel represents the worst streamlines extracted from the figure on the right hand side of the panel, showing the local compression error (see color code on the top of the figure) for the 8-bits (a) and 16-bits (b) octahedral quantization. The dataset was computed using the deterministic algorithm and $\delta = 1mm$.

Table 4: Compression ratios of `qfib` and `zfib`. The N/A values are the ones for which the algorithm was not able to perform the compression and decompression.

Stepsize δ		0.1 mm		0.2 mm		0.5 mm	
Nb. of streamlines		500k	3M	500k	3M	500k	3M
Compression ratios (in percentage)							
Method	Parameter	Deterministic					
`qfib`	8 bits	91.4	91.4	91.1	91.1	90.4	90.4
	16 bits	83.1	83.1	82.8	82.8	82.3	82.2
`zfib`	same*	N/A	N/A	78.4	N/A	96.6	96.8
	0.2 mm	98.1	98.1	95.9	96.0	87.5	87.5
Method	Parameter	Probabilistic					
`qfib`	8 bits	91.4	91.4	91.2	91.2	90.8	90.8
	16 bits	83.1	83.1	82.9	82.9	82.6	82.6
`zfib`	same*	N/A	N/A	78.1	N/A	87.1	N/A
	0.2 mm	96.0	N/A	88.7	N/A	69.9	N/A

same*: same error than `qfib` when using an 8 bit octahedral quantization (Table.3).

Table 5: Computation times of `qfib` and `zfib`. With `zfib`, 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.

Stepsize δ		0.1 mm		0.2 mm		0.5 mm	
Nb. of streamlines		500k	3M	500k	3M	500k	3M
Compression time (s)							
Deterministic	`qfib` (fibo)	24.1	144	12.8	74.8	5.49	32.0
	`qfib` (octa)	7.83	46.5	3.81	22.6	1.67	9.76
	`zfib`	702	4243	387	2284	387	2373
Probabilistic	`qfib` (fibo)	27.9	167	15.3	90.6	8.27	49.7
	`qfib` (octa)	8.61	54.7	4.86	29.0	2.53	15.5
	`zfib`	910	N/A	1052	N/A	1418	N/A
Decompression time (s)							
Deterministic	`qfib` (fibo)	4.98	30.1	2.61	15.4	1.14	6.79
	`qfib` (octa)	3.56	20.7	1.90	11.3	0.88	5.30
	`zfib`	12.1	72.7	12.9	77.1	17.3	103
Probabilistic	`qfib` (fibo)	5.77	34.9	3.23	18.9	1.75	10.3
	`qfib` (octa)	4.08	24.3	2.24	13.3	1.29	7.47
	`zfib`	28.5	N/A	43.0	N/A	60.8	N/A

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 `qfib` and `zfib`. 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 error of `zfib` to 0.2 mm as it is the default value on the provided source code. We can see that, for both compression and decompression, our method is at least two orders of magnitude faster than `zfib`. Moreover, during our experiments using the source code provided by the authors, `zfib` failed to compress very large files whereas our method is more scalable.

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 $0.2mm$. 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 3.

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)).

Stepsize δ		0.1 mm		0.2 mm		0.5 mm	
Nb. of streamlines		500k	3M	500k	3M	500k	3M
Average error of FA (in percentage)							
Method	Bits	Deterministic					
`qfib` fibonacci	5	0.004	0.004	0.008	0.009	0.023	0.023
	16	0.000	0.000	0.000	0.000	0.000	0.000
`qfib` octahedral	8	0.006	0.006	0.012	0.012	0.033	0.036
	16	0.000	0.000	0.000	0.000	0.001	0.000
`zfib`	-	2.506	2.527	2.572	2.553	2.200	2.219
Method	Bits	Probabilistic					
`qfib` fibonacci	8	0.002	0.002	0.004	0.002	0.003	0.002
	16	0.000	0.000	0.000	0.000	0.000	0.000
`qfib` octahedral	8	0.003	0.003	0.005	0.003	0.004	0.006
	16	0.000	0.000	0.000	0.000	0.000	0.000
`zfib`	-	0.575	N/A	0.347	N/A	0.077	N/A

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

Stepsize δ		0.1 mm		0.2 mm		0.5 mm	
Nb. of streamlines		500k	3M	500k	3M	500k	3M
Maximum error (absolute value $\times 10^{-2}$)							
Quantization	Precision	Deterministic					
Fibonacci	8 bits	3.91	3.93	7.92	8.32	22.6	22.9
	16 bits	0.29	0.29	0.16	0.15	0.19	0.23
Octahedral	8 bits	5.89	6.23	12.0	12.4	32.7	32.5
	16 bits	0.30	0.29	0.15	0.15	0.25	0.28
Quantization	Precision	Probabilistic					
Fibonacci	8 bits	2.43	2.25	3.66	3.93	11.1	12.4
	16 bits	0.09	0.09	0.09	0.09	0.44	0.48
Octahedral	8 bits	3.48	3.85	5.61	6.39	17.6	17.8
	16 bits	0.09	0.09	0.10	0.10	0.48	0.57
Average error (absolute value $\times 10^{-3}$)							
Quantization	Precision	Deterministic					
Fibonacci	8 bits	0.45	0.37	0.99	0.88	2.69	2.61
	16 bits	0.01	0.01	0.01	0.01	0.04	0.03
Octahedral	8 bits	0.67	0.53	1.45	1.31	3.95	3.83
	16 bits	0.01	0.01	0.01	0.01	0.04	0.04
Quantization	Precision	Probabilistic					
Fibonacci	8 bits	0.44	0.35	0.70	0.63	1.87	1.78
	16 bits	0.01	0.00	0.01	0.01	0.07	0.06
Octahedral	8 bits	0.69	0.54	1.12	0.99	2.77	2.63
	16 bits	0.01	0.01	0.01	0.01	0.08	0.07

3.5 Out-of-Core version.

To push forward the scalability of our method, we developed an out-of-core version of our algorithm. Compression times are presented in Table 8. They are higher than the in-core version. This is explained by the need to use the hard drive during compression and decompression in contrast to Table 5. This algorithm requires a few MB of RAM to run, no matter the tractogram size, making it possible to be used on any computer. To demonstrate that our algorithm is not limited to powerful hardware or small files, we decided to compress a huge dataset using a single-board computer. We generated a dataset containing 10 millions of streamlines, with a stepsize of $0.1mm$ using the probabilistic method, resulting in an 87.3GB tck file. For the hardware part, we used a Raspberry Pi 2 ($\approx$ 35\$) with 1GB of RAM, and a processor with 4 cores at 900 MHz. The data was stored in an external hard drive connected to the Raspberry Pi using the USB 2.0 interface. The compression resulted in a 7.49GB file with a compression ratio of 91.4% using an 8 bits octahedral quantization. It took around 4 hours, whereas the decompression took around 5.3 hours. These timings are consequent because of the USB 2.0 interface limitation. An important amount of time was indeed required to read and write all the data. This explains why the decompression took more time than the compression, as writing is slower than reading on a hard drive.

Table 8: Compression times of our out-of-core algorithm (`qfib`) with an 8 bits octahedral quantization.

Stepsize δ	0.1 mm		0.2 mm		0.5 mm	
Nb. of streamlines	500k	3M	500k	3M	500k	3M
Compression time (s)						
Deterministic	29.1	166	14.3	88.3	6.82	40.8
Probabilistic	33.0	192	17.9	103	9.99	57.9

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 GitHub[2].

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 ALLEGORI, 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.

[2] `https://github.com/syrousseau/qfib`

References

Alexandroni G, Zimmerman Moreno G, Sochen N, Greenspan H (2017) The fiber-density-coreset for redundancy reduction in huge fiber-sets. NeuroImage 146:246–256, DOI 10.1016/j.neuroimage.2016.11.027

Caiafa CF, Pestilli F (2017) Multidimensional encoding of brain connectomes. Scientific Reports 7(1):11491, DOI 10.1038/s41598-017-09250-w

Chung MK, Adluru N, Lee JE, Lazar M, Lainhart JE, Alexander AL (2009) Efficient parametric encoding scheme for white matter fiber bundles. IEEE EMBS 2009:6644–6647, DOI 10.1109/IEMBS.2009.5332866

Cigolle ZH, Donow S, Evangelakos D, Mara M, McGuire M, Meyer Q (2014) A Survey of Efficient Representations for Independent Unit Vectors. JCGT 3(2):1–30

Demir A, ¸Cetingül HE (2015) Sequential Hierarchical Agglomerative Clustering of White Matter Fiber Pathways. IEEE Trans Biomed Eng 62(6):1478–1489, DOI 10.1109/TBME.2015.2391913

Garyfallidis E, Brett M, Correia MM, Williams GB, Nimmo-Smith I (2012) QuickBundles, a Method for Tractography Simplification. Frontiers in Neuroscience 6(175)

González A (2010) Measurement of Areas on a Sphere Using Fibonacci and Latitude-Longitude Lattices. Mathematical Geosciences 42(1):49–64, DOI 10.1007/s11004-009-9257-x

Gori P, Colliot O, Marrakchi-Kacem L, Worbe Y, Fallani FDV, Chavez M, Poupon C, Hartmann A, Ayache N, Durrleman S (2016) Parsimonious Approximation of Streamline Trajectories in White Matter Fiber Bundles. IEEE Trans Med Imag 35(12):2609–2619, DOI 10.1109/TMI.2016.2591080

Guevara P, Poupon C, Rivière D, Cointepas Y, Descoteaux M, Thirion B, Mangin JF (2011) Robust clustering of massive tractography datasets. NeuroImage 54(3):1975–1993

Keinert B, Innmann M, Sänger M, Stamminger M (2015) Spherical Fibonacci Mapping. ACM TOG 34(6):193:1–193:7

Kumar K, Desrosiers C (2016) A sparse coding approach for the efficient representation and segmentation of white matter fibers. In: IEEE ISBI, pp 915–919, DOI 10.1109/ISBI.2016.7493414

Lindstrom P (2014) Fixed-Rate Compressed Floating-Point Arrays. IEEE TVCG 20(12):2674–2683

Liu M, Vemuri BC, Deriche R (2012) Unsupervised automatic white matter fiber clustering using a Gaussian mixture model. IEEE International Symposium on Biomedical Imaging 2012(9):522–525, DOI 10.1109/ISBI.2012.6235600

Maddah M, Wells WM, Warfield SK, Westin CF, Grimson WEL (2007) Probabilistic Clustering and Quantitative Analysis of White Matter Fiber Tracts. In: IPMI, vol 20, pp 372–383

Mercier C, Gori P, Rohmer D, Cani MP, Boubekeur T, Thiery JM, Bloch I (2018) Progressive and Efficient Multi-Resolution Representations for Brain Tractograms. In: EG VCBM, pp 89–93

Meyer Q, Süßmuth J, Sußner G, Stamminger M, Greiner G (2010) On Floating-point Normal Vectors. In: EGSR, pp 1405–1409

Moreno GZ, Alexandroni G, Sochen N, Greenspan H (2017) Sparse Representation for White Matter Fiber Compression and Calculation of Inter-Fiber Similarity. In: Computational Diffusion MRI, pp 133–143

Olivetti E, Bertò G, Gori P, Sharmin N, Avesani P (2017) Comparison of Distances for Supervised Segmentation of White Matter Tractography. In: PRNI, pp 1–4, DOI 10.1109/PRNI.2017.7981502, 1708.01440

Petrovic V, Fallon J, Kuester F (2007) Visualizing Whole-Brain DTI Tractography with GPU-based Tuboids and LoD Management. IEEE TVCG 13(6):1488–1495, DOI 10.1109/TVCG.2007.70532

Presseau C, Jodoin PM, Houde JC, Descoteaux M (2015) A new compression format for fiber tracking datasets. NeuroImage 109:73 – 83

Rheault F, Houde JC, Descoteaux M (2017) Visualization, Interaction and Tractometry: Dealing with Millions of Streamlines from Diffusion MRI Tractography. Frontiers in Neuroinformatics 11:42

Rousseau S, Boubekeur T (2017) Fast Lossy Compression of 3D Unit Vector Sets. In: SIGGRAPH Asia Tech. Briefs, pp 23:1–23:4

Siless V, Chang K, Fischl B, Yendiki A (2018) AnatomiCuts: Hierarchical clustering of tractography streamlines based on anatomical similarity. NeuroImage 166(Supplement C):32–45, DOI 10.1016/j.neuroimage.2017.10.058

Soares J, Marques P, Alves V, Sousa N (2013) A hitchhiker's guide to diffusion tensor imaging. Frontiers in Neuroscience 7:31

Tournier JD, Mori S, Leemans A (2011) Diffusion tensor imaging and beyond. Magnetic Resonance in Medicine 65(6):1532–1556

Tournier JD, Calamante F, Connelly A (2012) MRtrix: Diffusion tractography in crossing fiber regions. Int J of Imaging Systems and Technology 22(1):53–66

Van Essen D, Ugurbil K, Auerbach E, et al. (2012) The Human Connectome Project: A data acquisition perspective. NeuroImage 62(4):2222–2231

Wassermann D, Bloy L, Kanterakis E, Verma R, Deriche R (2010) Unsupervised white matter fiber clustering and tract probability map generation: Applications of a Gaussian process framework for white matter fibers. NeuroImage 51(1):228–241, DOI 10.1016/j.neuroimage.2010.01.004

Zhang F, Wu Y, Norton I, Rigolo L, Rathi Y, Makris N, O'Donnell LJ (2018) An anatomically curated fiber clustering white matter atlas for consistent white matter tract parcellation across the lifespan. NeuroImage 179:429–447, DOI 10.1016/j.neuroimage.2018.06.027

Zhang S, Laidlaw DH (2002) Hierarchical Clustering of Streamtubes. Technical report, Brown University, CS Department p 3

A Pseudo-code

```
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`