Supplemental Material

TRAKO: Efficient Transmission of Tractography Data for Visualization

Daniel Haehn[1][0000−0001−9144−3461], Loraine Franke[1][0000−0001−8560−2729], Fan Zhang[3][0000−0002−5032−6039], Suheyla Cetin-Karayumak[3][0000−0002−3315−5821], Steve Pieper[2][0000−0003−4193−9578], Lauren J. O'Donnell[3][0000−0003−0197−7801], and Yogesh Rathi[3][0000−0002−9946−2314]

[1] University of Massachusetts Boston, {haehn,franke}@mpsych.org
[2] Isomics, Inc., pieper@isomics.com
[3] Harvard Medical School,
{fzhang,skarayumak,odonnell,yogesh}@bwh.harvard.edu

1 Installation and Usage

TRAKO

Trako compresses DTI streamlines from .vtp to smaller .tko files!

Installation as PyPI package (recommended, preferably in a virtualenv)

```
pip install trako
```

Usage

```
./trakofy -i DATA/example.vtp -o /tmp/test.tko
./untrakofy -i /tmp/test.tko -o /tmp/restored.vtp
./tkompare -a DATA/example.vtp -b /tmp/restored.vtp
```

Developer installation (comes with test data)

Please follow these steps with Miniconda or Anaconda installed:

```
# create environment
conda create --name TRAKO python=3.6
conda activate TRAKO

# get TRAKO
git clone git@github.com:haehn/TRAKO.git
cd TRAKO

python setup.py install
```

Fig. 1: TRAKO is available through the PyPI package index. After installation, TRAKO provides three commandline tools: *trakofy* compresses fiber tracts, *untrakofy* restores the compressed tracts, *tkompare* compares two tractography files and computes errors. A developer installation of TRAKO comes with test data. The current TRAKO version is 0.3.4.dev9p.

2 Additional Experiment: TRAKO vs. VTK vs. TRK

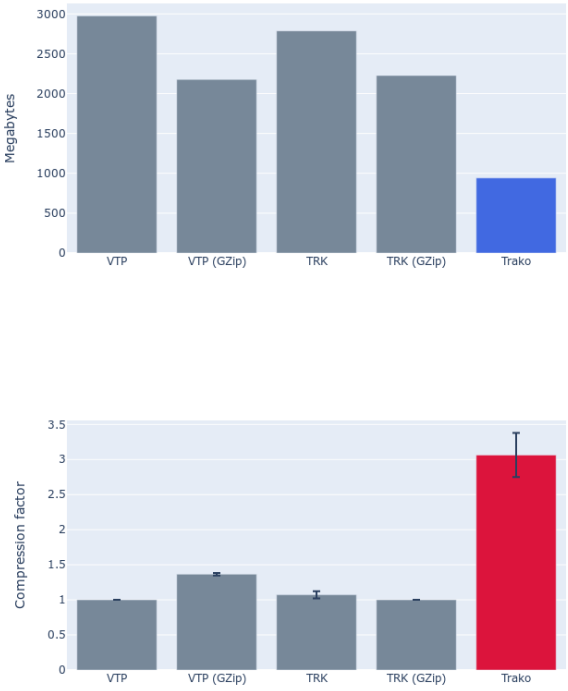

Fig. 2: As an additional experiment, we compared TRAKO and common streamline file formats that support per-vertex scalars and per-fiber properties (VTK/VTP[1], TrackVis TRK[2], plus general GZip[3] compression). We randomly chose two subjects of the ADHD data with 800 fiber clusters each. The combined input data in VTP format is roughly 3 Gigabytes. The data includes multiple per-fiber and per-vertex scalar values. TRAKO yields an average compression ratio of 3.2 and reduces the data size from 2974 Megabytes to 941 Megabytes.

[1] https://vtk.org/wp-content/uploads/2015/04/file-formats.pdf
[2] http://www.trackvis.org/dtk/?subsect=format
[3] https://www.gnu.org/software/gzip/

3 Configuration of TRAKO

3.1 Detailed Configuration for per-vertex and per-fiber values

TRAKO allows a detailed configuration of encoding parameters. Customizations can be configured in a JSON file to specify different parameters for different attributes. The example below configures a low bit rate of 7 for vertex INDICES (Integers). Due to little required precision, the INDICES do not need many bits to store a lossless representation. In contrast, a higher bitrate can be configured to map the large range of the RTOP scalar with sufficient precision. Existing compression methods do not support such detailed configurations. We integrated the OPTUNA framework[4] in TRAKO to allow exhaustive parameter exploration.

```
{

  'POSITION': {
    'position':True,
    'sequential':True,
    'quantization_bits':10,
    'compression_level':10,
    'quantization_range':-1,
    'quantization_origin':None
  },

  'INDICES': {
    'position':False,
    'sequential':True,
    'quantization_bits':7,
    'compression_level':10,
    'quantization_range':-1,
    'quantization_origin':None
  },

  'RTOP2': { # configure custom settings per attribute name
    'position':False,
    'sequential':True,
    'quantization_bits':16,
    'compression_level':10,
    'quantization_range':-1,
    'quantization_origin':None
  }

}
```

[4] https://optuna.org

3.2 Advanced Configuration Options

TRAKO (and the Draco pointcloud compression) supports the following advanced configuration options:

```
Position (True/False) treats points as X,Y,Z coordinates.

Sequential (True/False) keeps track of the order of points.
Should be always on for TRAKO.

Quantization bits should be an integer between 0 and 31

Compression level should be an integer between 0 and 10

Quantization_range is a float representing the size of the
bounding cube for the mesh. By default it is the range of
the dimension of the input vertices with greatest range.

Quantization_origin is the point in space where the bounding box
begins. By default it is a point where each coordinate is the
minimum of that coordinate among the input vertices.
```

Implementation note: We currently map all of these parameters through TRAKO's Cython interface[5] to Draco. This interface is a separate module and provides the C++ Draco implementation in Python. In addition to TRAKO, we also provide this Cython interface as open source software to allow the community to integrate Draco's compression algorithm in any Python project.

[5] `https://github.com/bostongfx/TrakoDracoPy`