zfp: Random-accessible compressed numerical arrays

EGU 2026

Peter Lindstrom

May 8, 2026

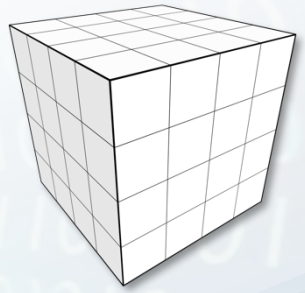
LLNL-CFPRES-2018308

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

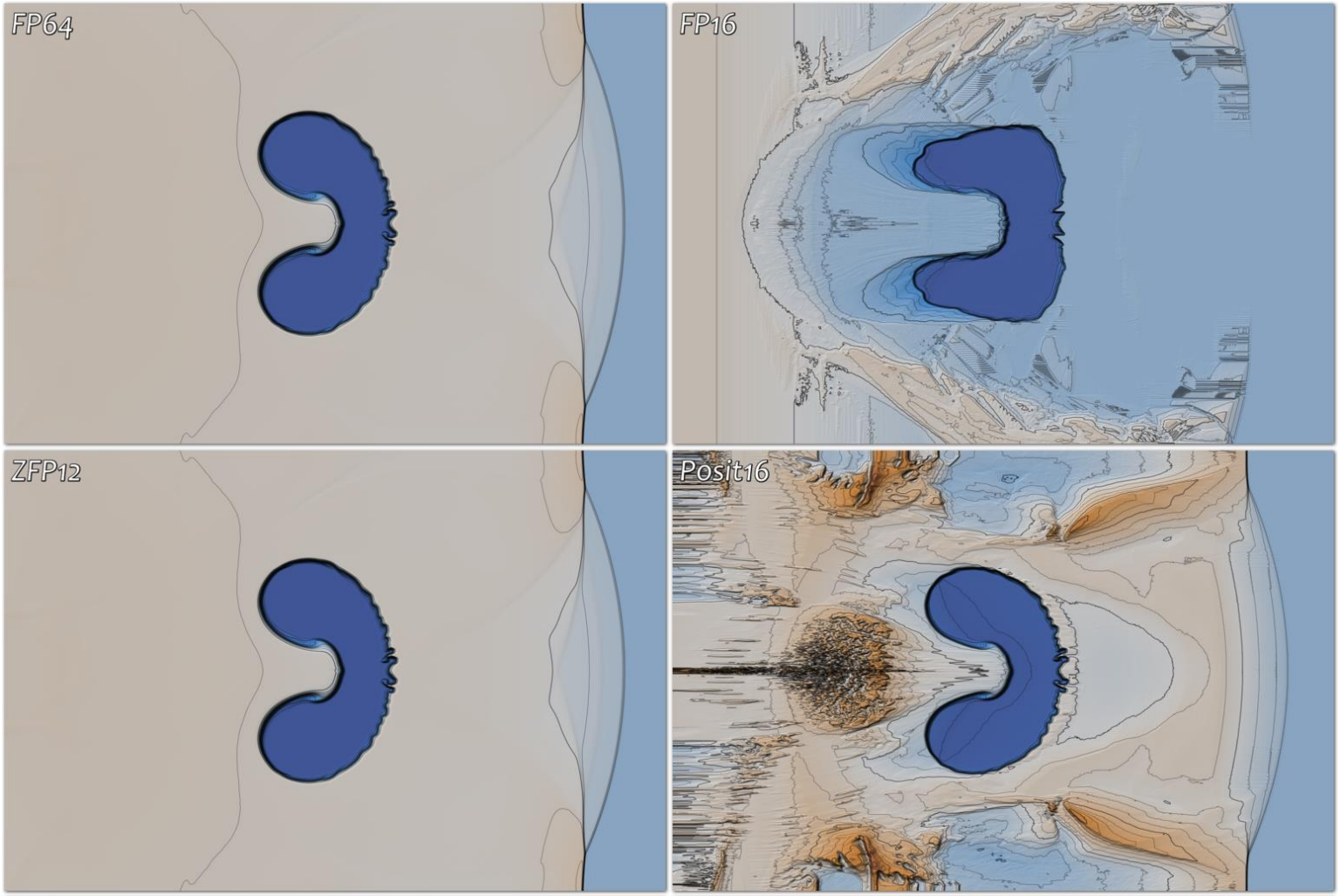
ZFP salient features

- Supports **integer** and **floating-point** multi-dimensional Cartesian data
- Supports **lossy** and **lossless** compression
- Supports **fixed rate**, absolute & relative **error tolerances**
- Supports **random access** reads from & writes to array elements
  - *d*-dimensional block of 4^d values is (de)compressed independently
  - `zfp::array2<double> a(nx, ny, rate)` ⟺ `std::vector<double> a(nx * ny)`



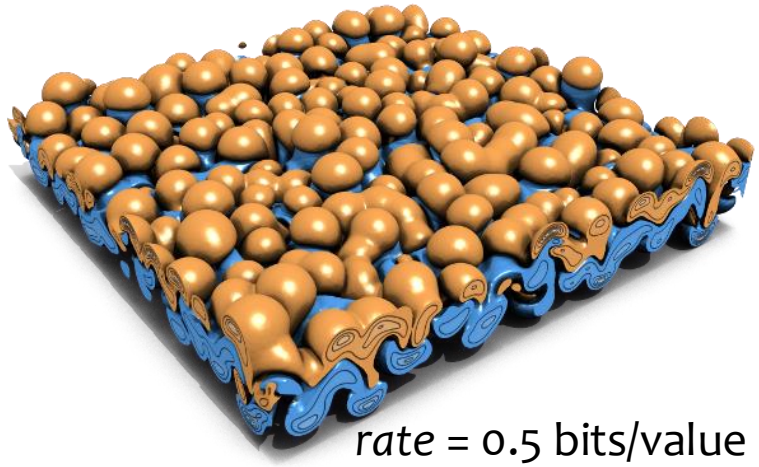
- Supports **massive data parallelism** through array partitioning into blocks
- Is **extremely fast** and runs on both CPUs and GPUs
- Was primarily designed for in-memory compressed storage
  - **Numerical computations** like PDEs, data analysis, visualization
  - Also **reduces data movement** in I/O, communication, CPU-GPU transfers

PDE computations using 12-bit ZFP are qualitatively identical to 64-bit IEEE double precision



Working with ZFP: compression modes and parameters

- ZFP supports four primary compression modes
  - **Fixed rate**: fix number of **compressed** bits/value
    - Allows specifying exact per-array storage budget
    - Enables **random access**
  - **Fixed precision**: fix number of **uncompressed** bits/value
    - Analogous to number of floating-point mantissa bits
    - Enables **relative error control**
  - **Fixed accuracy**: store enough bits to meet error tolerance
    - Original & decompressed values differ by no more than tolerance
    - Conservative error bound—actual error tends to be 2-8x smaller
    - Enables **absolute error control**
  - **Reversible**: bit-for-bit lossless compression
    - Compression ratio rarely exceeds 2x



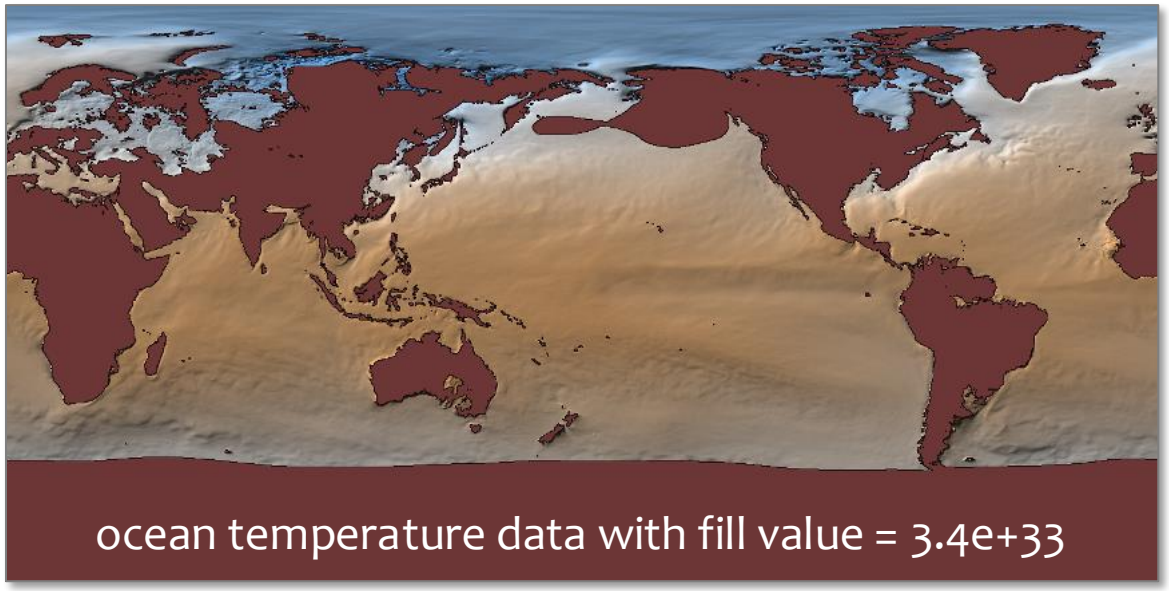
rate = 0.5 bits/value

ZFP supports multiple languages/back-ends and is widely used to reduce I/O, communication, in-memory storage

- Language support: **C, C++, Python, Fortran**
  - Julia, Rust, WebAssembly bindings through 3rd party
- Back-end support: **serial, OpenMP, CUDA, HIP**
  - AVX, SYCL, NEC VE implementations through 3rd party
  - FPGA, ASIC implementations available

ZFP is currently being extended to support fill values

- Naively compressing out-of-range fill values degrades accuracy & compression

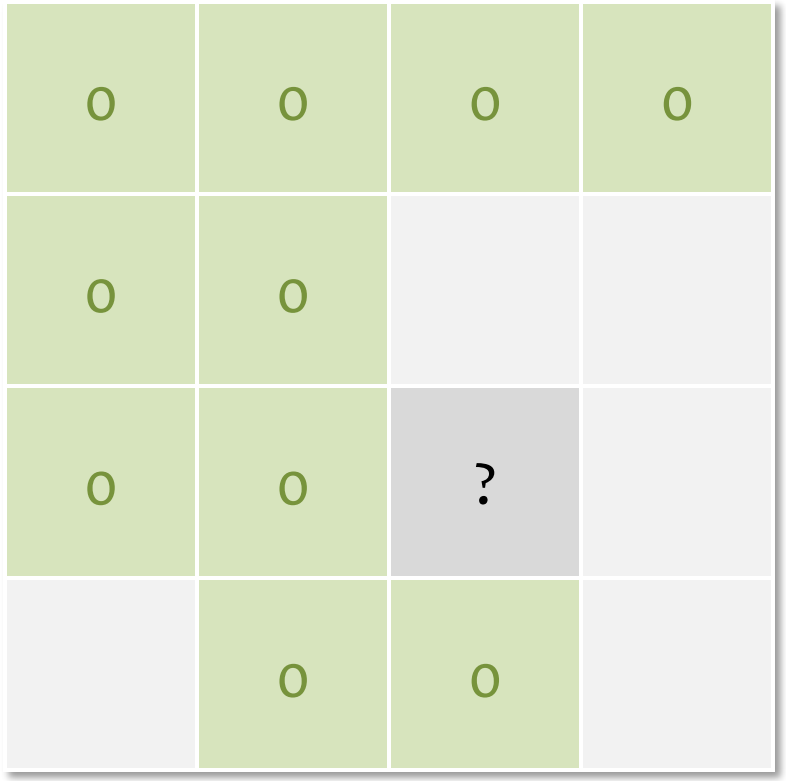


ocean temperature data with fill value = 3.4e+33

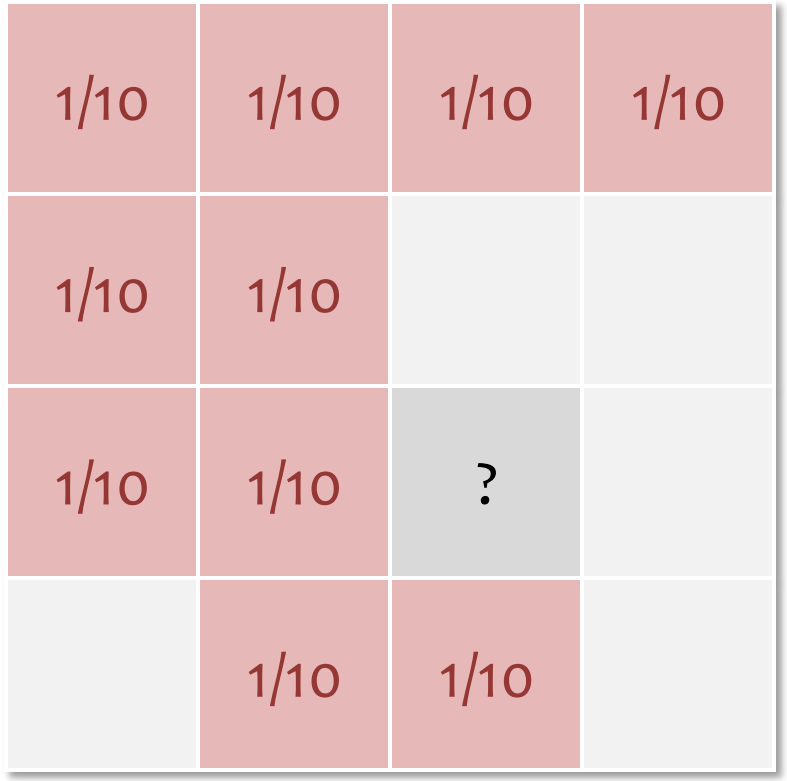
block artifacts appear where true & fill values meet

- Solution: compress bit mask losslessly, replace fill values with smooth interpolant
  - Express undefined values as weighted average of nearby defined values

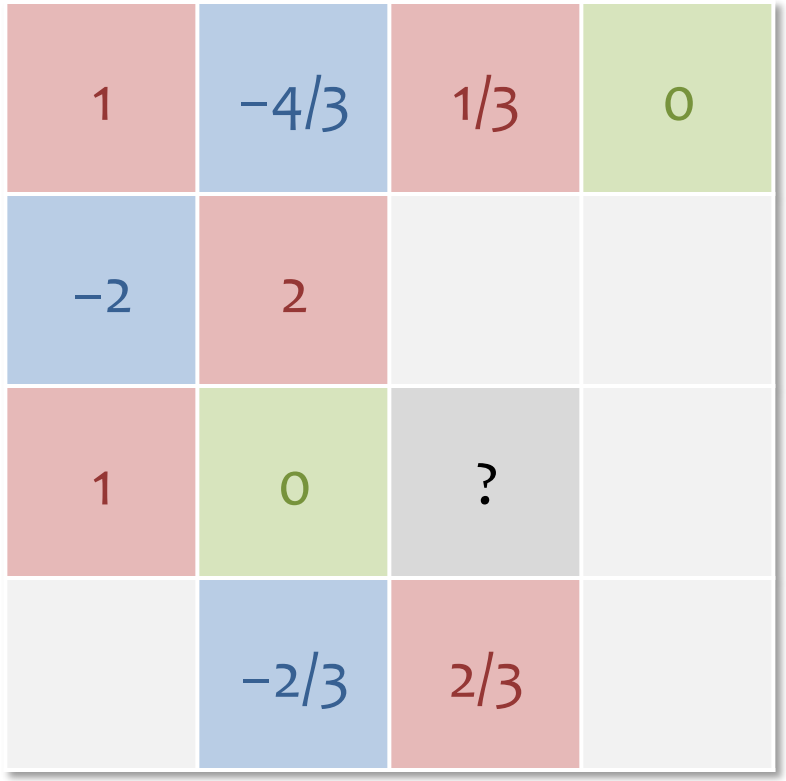
We have explored three weighted-average interpolants



zero interpolation
sets all weights & unknown values to zero

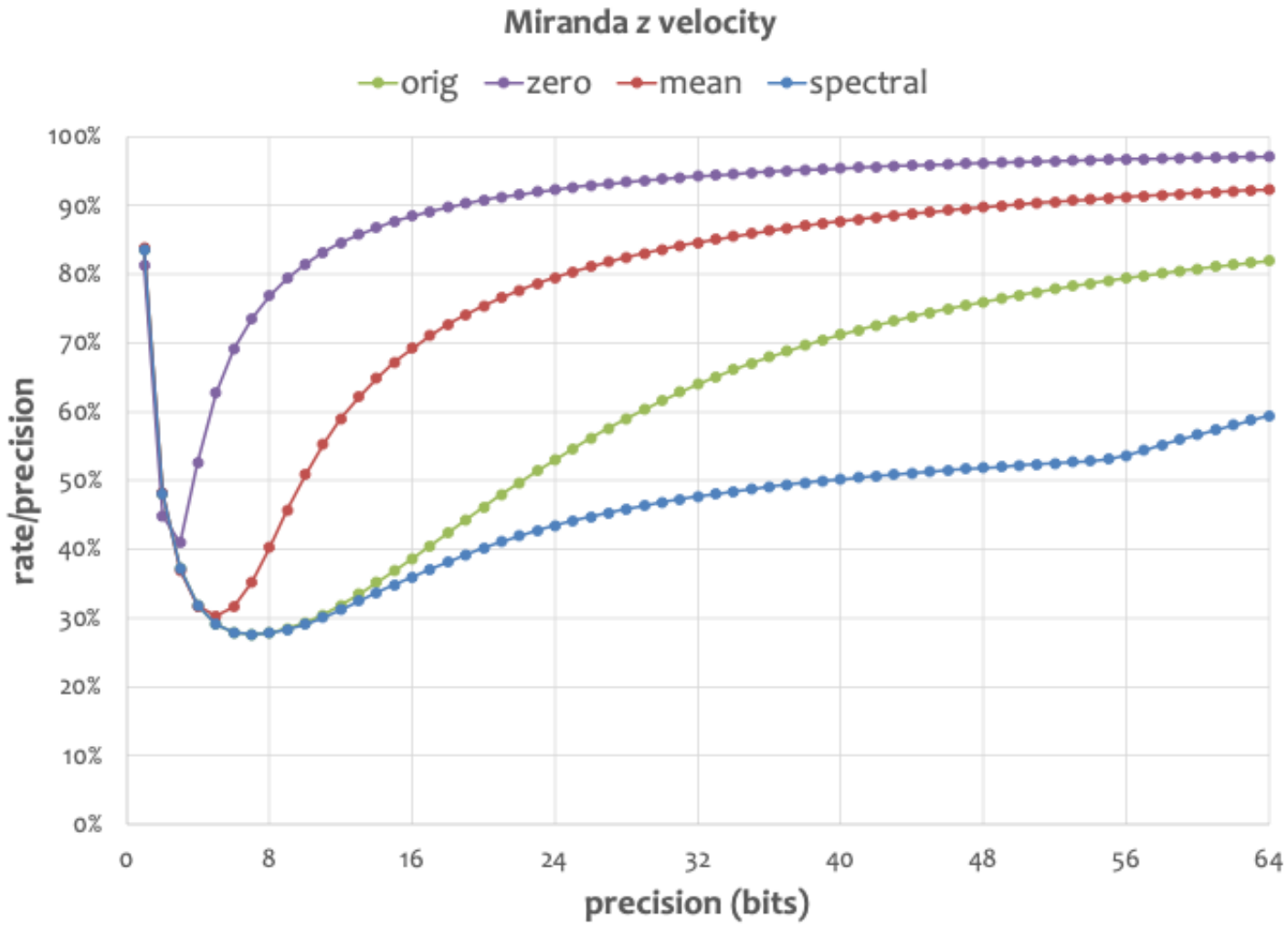


local mean interpolation
sets weights of all *n* known values to 1/*n*

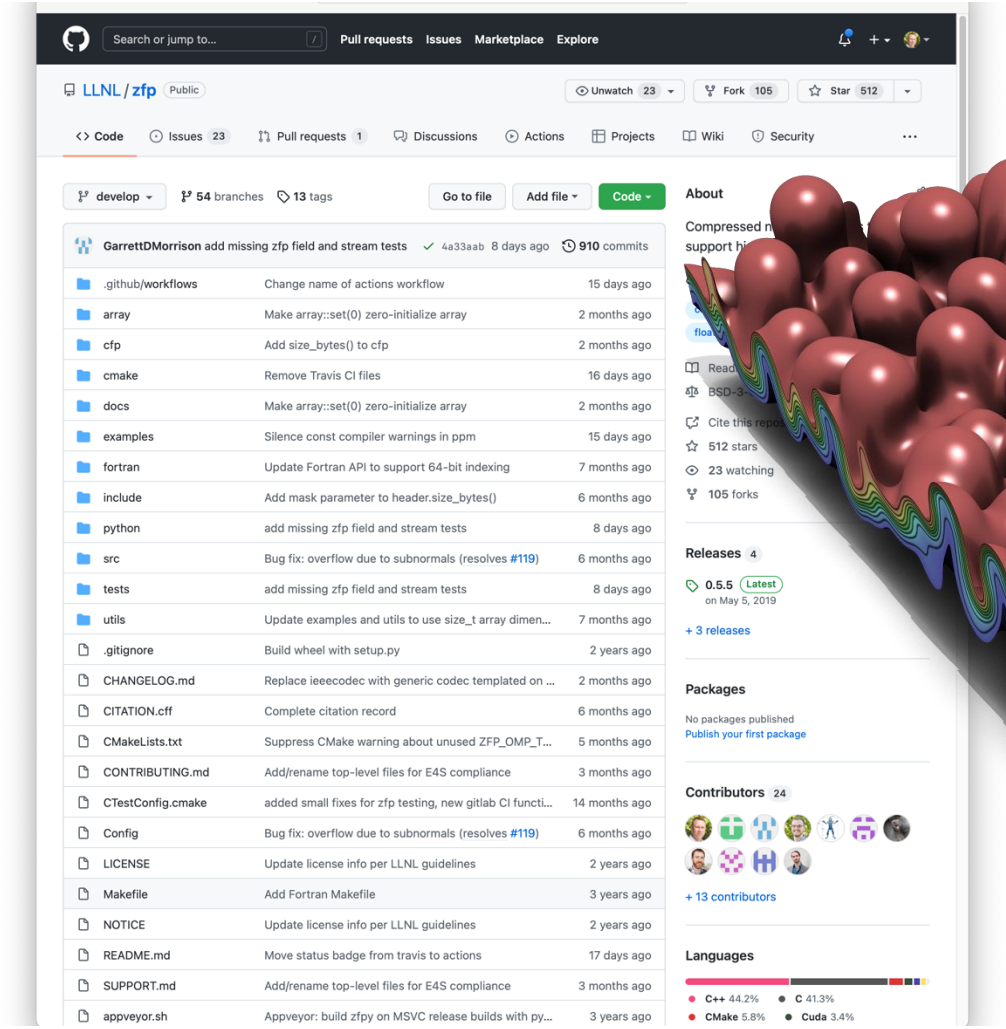


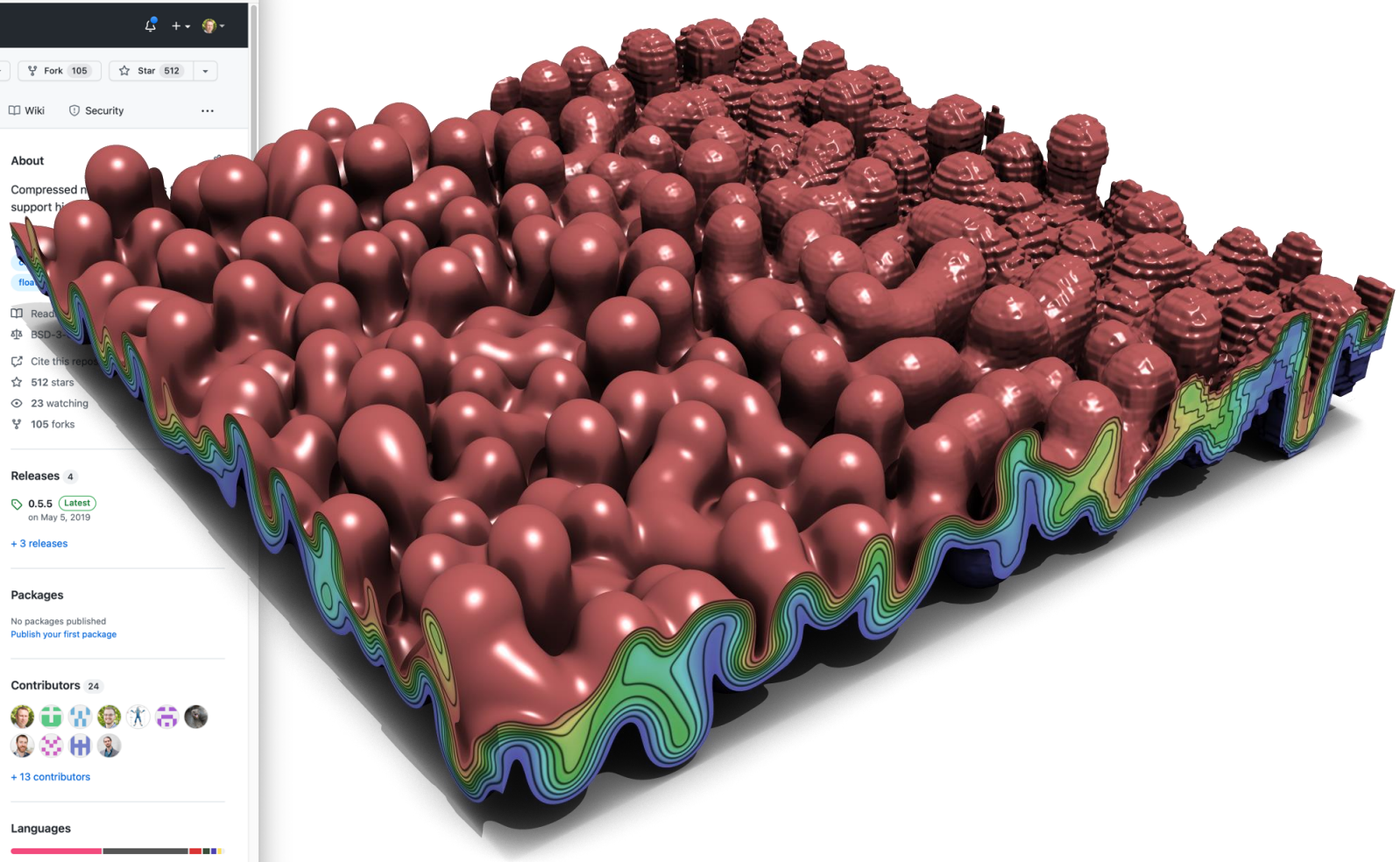
spectral interpolation
maximizes smoothness & compressibility
[Ibarria, Lindstrom, Rossignac DCC 2007]

Spectral interpolation of randomly removed data points greatly outperforms local-mean interpolation



ZFP is available as open source on GitHub: https://github.com/LLNL/zfp





Appendix: Installing ZFP 1.0.1 from GitHub

- Installation using gmake

```
$ git clone https://github.com/LLNL/zfp.git
$ cd zfp
$ make BUILD_EXAMPLES=1
$ make test                    # optional testing
```

- Installation using cmake

```
$ git clone https://github.com/LLNL/zfp.git
$ cd zfp
$ mkdir build
$ cd build
$ cmake -DBUILD_EXAMPLES=ON ..
$ cmake --build . --config Release
$ ctest                        # optional testing (use -DBUILD_TESTING_FULL for more tests)
```

Example: Lossless compression with the command-line tool

- Set path to example data (download from https://sdrbench.github.io)

```
$ export data=/example_data/Hurricane
```

- Reversible (lossless) compression mode (–R)

```
$ zfp -R -f -3 500 500 100 -i $data/Uf48.dat -h -z Uf48.zfp
type=float nx=500 ny=500 nz=100 nw=1 raw=100000000 zfp=65011064 ratio=1.54 rate=20.8
```

NOTE: –z <file> is optional (ZFP compresses to memory when omitted)

- Decompress and verify

```
$ zfp -h -z Uf48.zfp -o Uf48.dat
type=float nx=500 ny=500 nz=100 nw=1 raw=100000000 zfp=65011064 ratio=1.54 rate=20.8

$ diff -s $data/Uf48.dat Uf48.dat
Files /example_data/Hurricane/Uf48.dat and Uf48.dat are identical
```

Example: Lossy compression with the command-line tool

- Fixed-rate mode (–r) sets exact storage size in compressed bits/value

```
$ zfp -r 5 -f -3 500 500 100 -i $data/Uf48.dat -s
type=float nx=500 ny=500 nz=100 nw=1 raw=100000000 zfp=15625000 ratio=6.4
rate=5 rmse=0.017 nrmse=0.0001837 maxe=3.023 psnr=68.70
```

- Fixed-accuracy mode (–a) sets absolute error tolerance

```
$ zfp -a 0.1 -f -3 500 500 100 -i $data/Uf48.dat -s
type=float nx=500 ny=500 nz=100 nw=1 raw=100000000 zfp=15605160 ratio=6.41
rate=4.994 rmse=0.00251 nrmse=2.711e-05 maxe=0.01701 psnr=85.32
```

- "Accidentally" transpose data set dimensions (**be careful!**)

```
$ zfp -a 0.1 -f -3 100 500 500 -i $data/Uf48.dat -s
type=float nx=100 ny=500 nz=500 nw=1 raw=100000000 zfp=36601304 ratio=2.73
rate=11.71 rmse=0.00273 nrmse=2.949e-05 maxe=0.01817 psnr=84.59
```

Example: Parallel compression using OpenMP & CUDA

- Serial compression

  $ zfp -r 4 -f -3 500 500 100 -i $data/Uf48.dat # optional: –x serial

- OpenMP compression

  $ zfp -r 4 -f -3 500 500 100 -i $data/Uf48.dat **-x omp** # *N* threads: –x omp=*N*

- CUDA compression (**fixed-rate mode only**)

  $ zfp -r 4 -f -3 500 500 100 -i $data/Uf48.dat **-x cuda**