

Scientific Data Visualization Project Final Report

Jinyang Ruan, Tianhao Huang & Ricardo Cahue
December 13, 2021

Abstract

In scientific research, it is often necessary to use large data sets as the basis for experimental design. However, using large data sets may also bring inefficiencies and incomprehensibility. Proper compression before visualization of very large data sets contributes to a better understanding of the data sets and can greatly improve research efficiency. This project aims to compare the compression effect of two different compressors, SZ and ZFP, on the same data set. The compressed file is then visualized using VTK. We used PSNR to measure the distortion of the data set before and after compression.

Keywords: *data visualization, data compression, VTK*

1 Introduction

Interpreting hidden information in multi-dimensional data is a challenging and complex task. Generally, data compression is considered the first step in multi-dimensional data analysis and exploration. Although the compression algorithm can greatly reduce the complexity of the multidimensional data sets, it cannot help users better understand the data. On the contrary, an inappropriate algorithm may weaken some features of the original data set, making the compressed data set more difficult to understand. As a result, visualization tools become the first choice to help intuitively understand data sets.

This project first uses VTK to visualize a cube. Then different compression algorithms are used to compress and decompress the data. Then visualize the results again. After that, compare the visualizations of the data sets before and after compression. Finally, PSNR value is used to measure the distortion of the compressed dataset.

2 Problem Definition

The purpose of this project is to visualize the pre - and post-compression data sets separately using VTK and compare the influence of SZ and ZFP on the original data set. Then, the distortion of the dataset before and after compression is evaluated by calculating PSNR. Finally, under the condition of maintaining a similar compression ratio, the algorithm with better performance is found according to the PSNR value.

3 Tools and Measures

3.1 Dataset

To accomplish our goal, we first need to have a rough understanding of the structure of the original data set. We use the TeraShake 2.1 earthquake simulation data set [1]. This data set is time-varying velocity vector data. All simulation data is big-endian, 4 byte, IEEE floating point binary format data. The data values are velocity in meters/sec. The velocities created by the simulation are broken into 3 components (x,y,z) and are divided into files by component and time step. Although the description file states that each time step contains three files (x, y, z), after preliminary compression, we find that each of the three files contains complete 3D data (x, y, z). Therefore, the names of data files used in the demonstration are only including "X", but we are actually computing the complete 3D data. The volumetric data files are 750x375x100 in space and a total of 227 time steps. These files are named TS21z_Component_R2_Timestep.bin (for example, TS21z_X_R2_008000.bin corresponds to time-step 80 seconds for the velocity data) [1].

3.2 SZ

SZ is an error-bounded HPC in-situ data compressor for significantly reducing the data sizes, which can be leveraged to improve the checkpoint/restart performance and post-processing efficiency for HPC executions [2][3][4]. SZ can be used to compress different types of data (single-precision and double-precision) and any shapes of the array. Higher dimensions can also be extended easily [2][3][4].

SZ allows setting the compression error bound based on absolute error bound and/or relative error bound, or point-wise relative error bound, by using sz.config [2][3][4]. We will select the appropriate method from the following compression error boundary to compress the original data set:

- Absolute error bound (namely absErrBound in the configuration file sz.config): It is to limit the compression and decompression errors to be within an absolute error [2].
- Relative error bound (called relBoundRatio in the configuration file sz.config): It is to limit the compression and decompression errors by considering the global data value range size (i.e., taking into account the range size (max_value - min_value)) [2].
- Point-wise relative error bound: It is to control the compression errors based on a relative error ratio in comparison with each data point's value [2].

3.3 ZFP

ZFP is an open-source library for compressed numerical arrays that support high throughput read and write random access [5]. zfp also supports streaming compression of integer and floating-point data. To achieve high compression ratios, zfp generally uses lossy but optionally error-bounded compression. Bit-for-bit lossless compression is also possible through one of zfp's

compression modes.zfp works best for 2D and 3D arrays that exhibit spatial correlation [5], such as continuous fields from physics simulations, images, regularly sampled terrain surfaces, etc.

ZFP compresses d-dimensional (1D, 2D, 3D, and 4D) arrays of integer or floating-point values by partitioning the array into cubical blocks of 4^d values, i.e., 4, 16, 64, or 256 values for 1D, 2D, 3D, and 4D arrays, respectively [5]. Each such block is compressed independently into a fixed- or variable-length bit string, and these bit strings are concatenated into a single stream of bits [5][6]. We also choose the ZFP algorithm with different compression error boundaries to compress the same data set. Through the comparison with the results of the SZ algorithm to find out the algorithm with better performance:

- *Lossless mode*: no information is expected to loss during the compression and decompression process. We tested this mode for observing the compression ratio, but we are not going to visualize it since it is considered exactly the same as the original data.
- *Fixed-rate mode*: each d-dimensional compressed block of 4^d values is stored using a fixed number of bits. This number of compressed bits per block is amortized over the 4^d values. The fixed-rate mode also ensures a predictable memory/storage footprint but usually results in far worse accuracy per bit than the variable-rate fixed-precision and fixed-accuracy modes [6].
- *Fixed accuracy*: all transform coefficient bit planes up to a minimum bit plane number are encoded. As in fixed-precision mode, the number of bits used per block is not fixed but is dictated by the data. Use *tolerance* = 0 to achieve near-lossless compression. Fixed-accuracy mode gives the highest quality (in terms of absolute error) for a given compression rate and is preferable when random access is not needed [6].

3.4 VTK

We chose to use VTK for our data visualization work, according to the instructions of the website that provided the data set. VTK is an open-source software system for image processing, 3D graphics, volume rendering, and visualization. VTK includes many advanced algorithms and rendering techniques.

Before using VTK for visualization, we need to create two Tcl script files. The first script converts a file in the dataset to a VTK format dataset. The second script loads this data into VTK and displays it using the image viewer.

3.5 PSNR

The Peak signal-to-noise ratio (PSNR) computes the peak signal-to-noise ratio, in decibels, between two images [8]. This value of PSNR can be used to quantify the quality of compressed or reconstruct data. The higher the PSNR value, the better the quality of the compressed, or reconstructed image.

The mean-square error (MSE) and the peak signal-to-noise ratio (PSNR) is used to compare image compression quality [8]. The MSE represents the cumulative squared error between the

compressed and the original image, whereas PSNR represents a measure of the peak error. The lower the value of MSE, the lower the error. For computing the MSE value and PSNR value, we used the following equations [8]:

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2$$

$$PSNR = 10 * \log_{10} \left(\frac{MAX_I^2}{MSE} \right) = 20 * \log_{10} \left(\frac{MAX_I}{\sqrt{MSE}} \right)$$

Typical values for the PSNR in the lossy image and video compression are between 30 and 50 dB, provided the bit depth is 8 bits, the higher the better [8]. Values over around 50 means the compressed data almost same as the original data; values over 30 means there are some differences between compressed data and original data, but the differences can hardly be observed by naked eyes; values below 30 means the compressed data get distorted and the differences can be easily observed by naked eyes.

4 Implementation

4.1 Original data visualization

The first step of the project is to visualize the original data. The data files we downloaded and decompressed from the official website are in binary format. In this case, Tcl scripts are used to convert the binary file to the VTK format so that we can use VTK to implement visualization. Note that we need to put the script and the target files on the same path. A clean instance of Tcl script for converting binary data to VTK readable format can be shown in the figure below. See Appendix A for the complete codes.

```
package require vtk
vtkImageReader reader
    reader SetFileName "FileName.bin"
...
reader SetDataExtent 0 749 0 374 0 99
...
vtkStructuredPointsWriter writer
    writer SetInput [reslice1 GetOutput]
    writer SetFileName "FileName.bin.vtk"
...
exit
```

Figure 1. A Tcl script for converting a binary data file to vtk format

As the figure shows, the vtk package is required. We manually set the data dimension to 3 and also set the data structure as a 750*375*100 array. For a better comparison, we reslice the input 750*375*100 file along with the Z-axis into 100 2D files with the 750*375*1 structure.

After converting the original binary data to the VTK readable format, we used Tcl script again for visualization. A part of the script can be shown in figure 2 below. The full script can be seen in Appendix B.

```
package require vtk
package require vtkinteraction
vtkImageReader reader
...
reader SetDataExtent 0 749 0 374 0 99
reader SetDataOrigin 0 0 0
reader SetDataSpacing 800 800 800
reader Update
scan [[reader GetOutput] GetWholeExtent] "%d %d %d %d %d %d" \
xMin xMax yMin yMax zMin zMax
...
toplevel .top
wm protocol .top WM_DELETE_WINDOW ::vtk::cb_exit
...
scale .top.slice \
    -from $zMin \
    -to $zMax \
    -orient horizontal \
    -command SetSlice \
    -variable slice_number \
    -label "Z Slice"
...
pack $vtkiw \
    -side left -anchor n \
    -padx 3 -pady 3 \
    -fill x -expand f
...
    -fill x -expand f
tkwait window .
```

Figure 2. A Tcl script for visualizing vtk file

Note that for visualization we need VTK and vtkinteraction packages. We also need to manually put the input file structure (750*375*100) to the reader. One example visualization result can be shown as follows. For every visualization file, we can see 100 images on one screen by dragging the z-axis points.

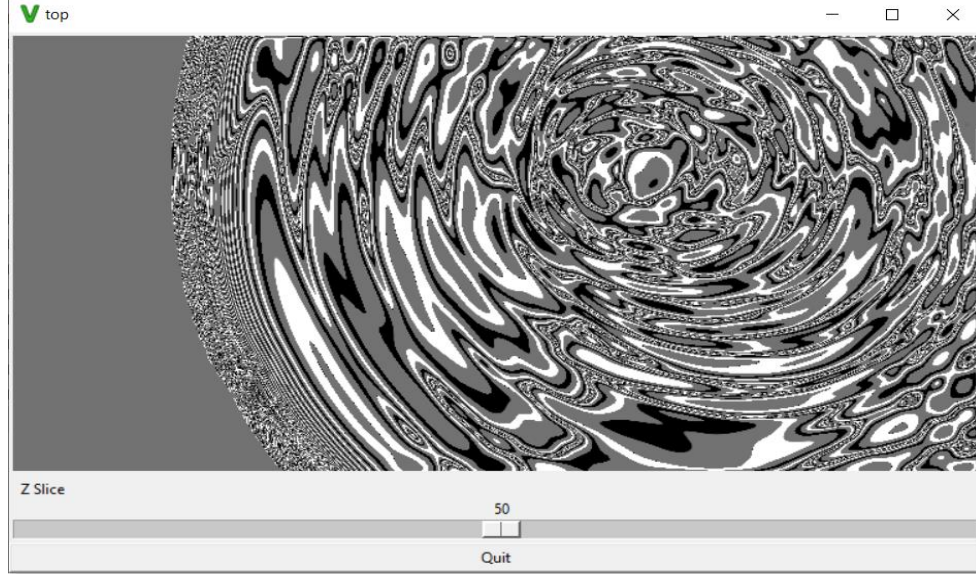


Figure 3. TS21z_X_R2_004000.vtk visualization at the 50th Z-axis point

4.2 Original data compression

After visualizing the original data, we used two compressors SZ and ZFP. When using the SZ compressor, we chose to use the ABS (absolute norm) error-bound mode. When using the ZFP compressor, we chose to use lossless, fixed-accuracy, and fixed-rate error-bound modes for comparison. Note that the effective area of the picture is increasing over time since seismic waves diffuse over time, so the default SZ ABS error bound mode tends to be the low compression ratio. Thus, we need to change the fixed accuracy and fixed rate in ZFP to make the compression ratio roughly the same. The compression detailed information can be shown in the table below.

File ID	Original File Size (KB)	Compressed File Size (KB) Ratio (SZ ABS)	Compressed File Size (KB) Ratio (ZFP Lossless)	Compressed File Size (KB) Ratio (ZFP accuracy)	Compressed File Size (KB) Ratio (rate) (ZFP rate)
001000	109,864	12,859 8.54	14,061 7.81	11,726 9.37	13,807 7.96 (4)
002000	109,864	42,851 2.56	46,712 2.35	39,106 2.81	41,419 2.65 (12)
003000	109,864	67,392 1.63	73,773 1.49	61,629 1.78	62,129 1.77 (18)
004000	109,864	83,771 1.31	91,936 1.19	76,688 1.43	69,032 1.59 (20)
008000	109,864	102,436 1.07	112,489 0.977	94,420 1.16	103,574 1.06 (30)

Table 1. Detailed compression info

One interesting observation is that when compressing the 008000 data file, ZFP lossless mode makes the compressed data larger than original data, even if it will go back to its original size and keep all information (lossless) after decompressing it. The reason for this is that, first of all,

lossless compression needs to retain more of the original data characteristics than lossy compression. Only invertible operations can be used for correlation transformation during coding. At the same time, the accuracy of the original data should be guaranteed to achieve lossless requirements during coding. In addition, data files with a time step of 8000 hardly contain all zeros and too many duplicates. As a result, there is very little data to compress. And compression also needs to generate additional dictionaries, encoding format and other control information, so the size of the compressed file is larger than the original file.

4.3 Compressed data visualization

Specifically, we were not visualizing the compressed data since we were not able to know the data structure of the compressed data. The strategy we used is to decompress the compressed data using the same compress error-bound mode. For example, when using the ZFP compressor we used the fixed rate 4 to compress the 001000 file, but we still use the fixed rate 4 to decompress the compressed data, so that we can achieve the same size and structure of the data file. After decompressing all compressed data to the target data structure, we use the same Tcl scripts for visualization. Figure 4 shows the visualization result of the file which is compressed and decompressed from the file in figure 3 by the SZ compressor.

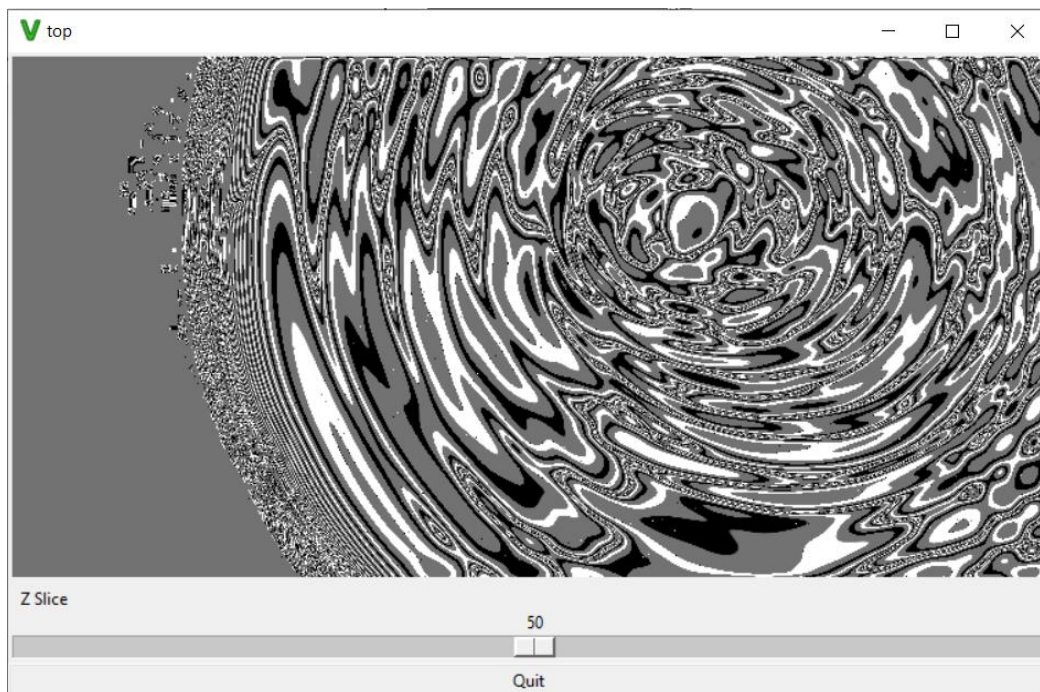


Figure 4. TS21z_X_R2_004000.sz.out.vtk visualization at the 50th Z-axis point

4.4 PSNR computation

The python code for computing PSNR between two binary files is shown in figure 5 below.

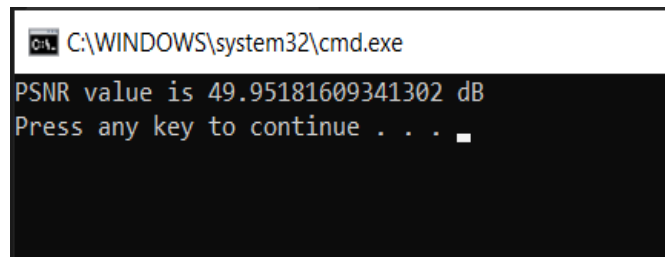
```

from math import log10, sqrt
import numpy as np
def PSNR(original, compressed):
    mse = np.mean((original - compressed) ** 2)
    if(mse == 0): # MSE is zero means no noise is present in the signal .
        # Therefore PSNR have no importance.
        return 100
    max_pixel = 255.0
    psnr = 20 * log10(max_pixel / sqrt(mse))
    return psnr
def main():
    original = np.fromfile('path\to\the\original\data.bin', np.uint8)
    compressed = np.fromfile('path\to\the\decompressed\data.out', np.uint8)
    value = PSNR(original, compressed)
    print(f'PSNR value is {value} dB')
if __name__ == "__main__":
    main()

```

Figure 5. Python code for computing PSNR value between two binary data files

As the code shows, the input file's bit depth is 8 bits (np.uint8), the PSNR value should be between 30 to 50, where higher is better [7]. The result of computing PSNR value between 004000 original file and 004000.sz.out file is shown in figure 6.



```

C:\WINDOWS\system32\cmd.exe
PSNR value is 49.95181609341302 dB
Press any key to continue . . .

```

Figure 6. PSNR value of TS21z_X_R2_004000.sz.out.vtk and the original data

49.95 is an excellent result which means there is almost nothing lost during the compression process. By observing figure 3 and figure 4, we can also make the same conclusion—only a little distortion in the upper left corner.

5 Results and Discussion

After implementing the steps in section 4 on 30 different files, we selected 3 files based on the different compression ratios that present interesting results.

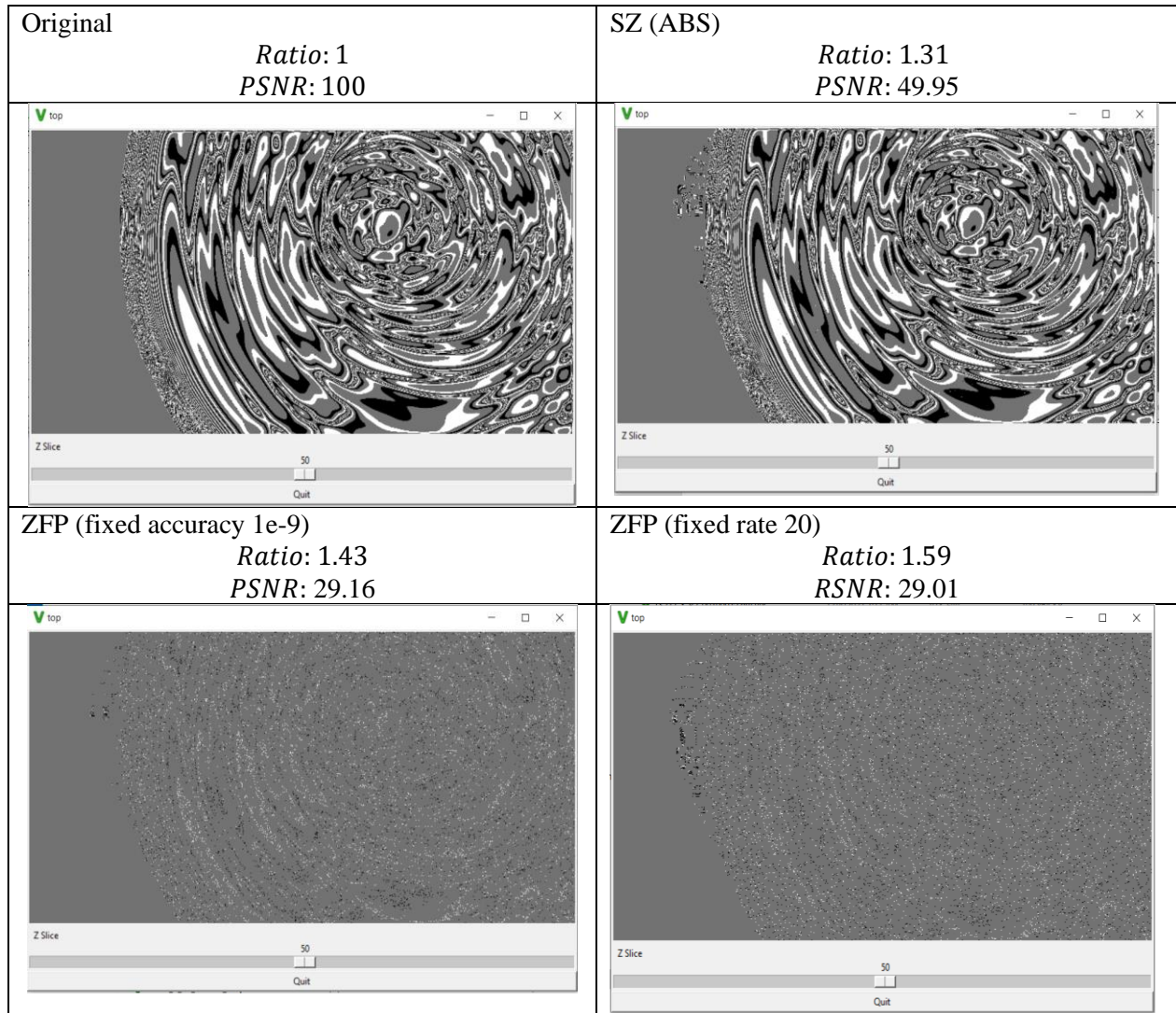


Figure 7. Visualization results on the 004000 file

Figure 7 shows the visualization results on the 004000 file which contains a little blank area. We managed the compression ratio around 1.5. From the visual perspective, we can easily conclude that SZ performed the best in this case. Almost all information has been kept, and details can still be observed. ZFP compressor for both fixed accuracy mode and fixed-rate mode, they only kept the outline of the image, detailed information was lost, but the blank area did not distort. To quantify the image quality, PSNR values presented the same conclusion. 49.95 is a very high score, it means the compressed data is almost exactly same as the original data, 29.16 and 29.01 are quite low since the standard range of the PSNR value is between 30 to 50, score under 30 means there are significant differences between compressed data and original data, and the differences can be easily observed by naked eyes.

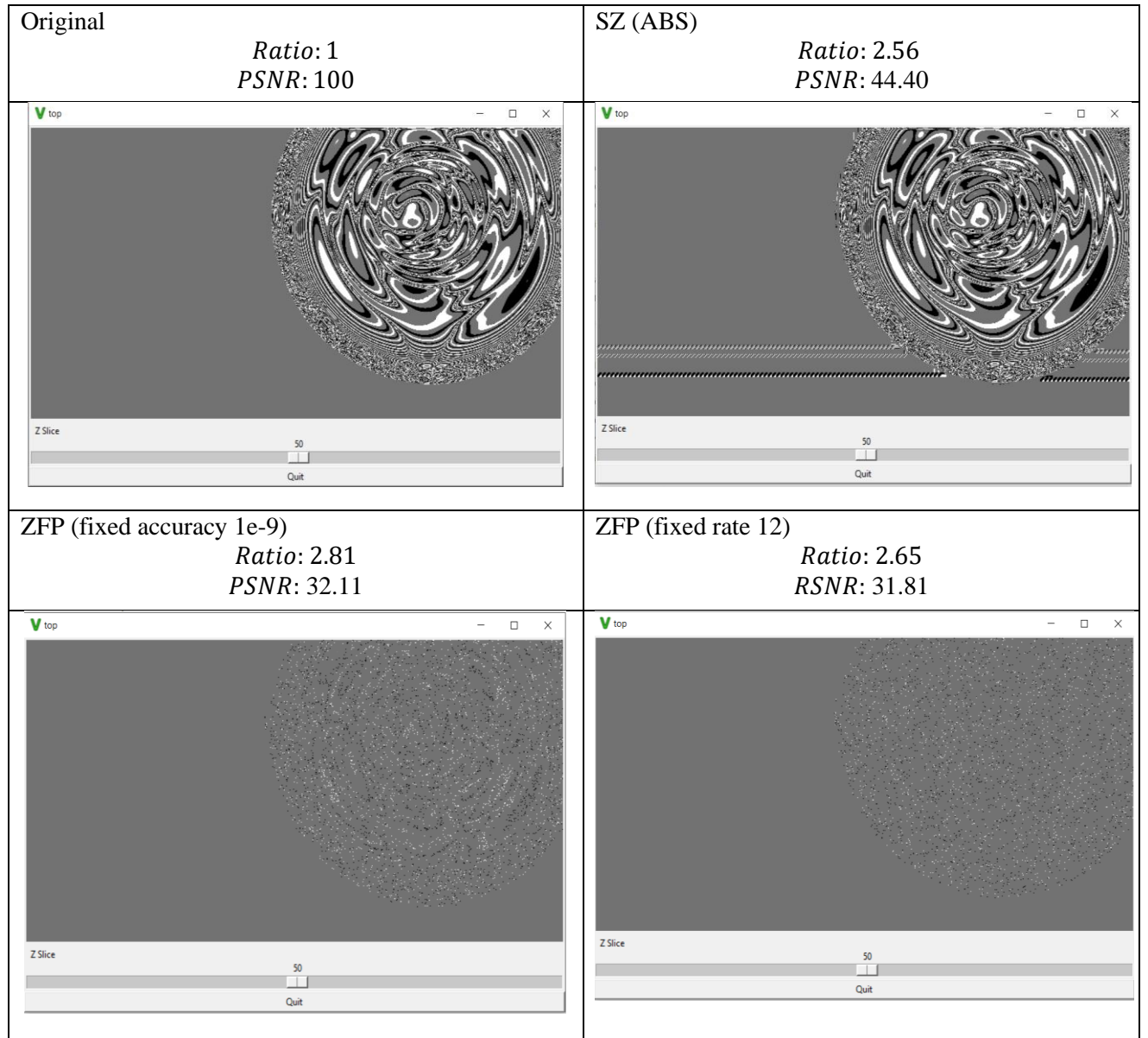


Figure 8. Visualization results on the 002000 file

Figure 8 shows the visualization results on the 002000 file which contains about half the blank area. From observation, we can conclude that SZ easily makes distortion on the blank area, but it performs well on the details. ZFP goes the opposite—there is not any distortion on the blank area, but it only keeps the outline of details. So as the increasing of the blank areas, the PSNR scores between ZFP compressed file and original file go up, and the PSNR scores between SZ compressed file and original file go down. To quantify the proportion of blank area, we used the compression ratio we got from SZ compressor with the default ABS error-bound mode.

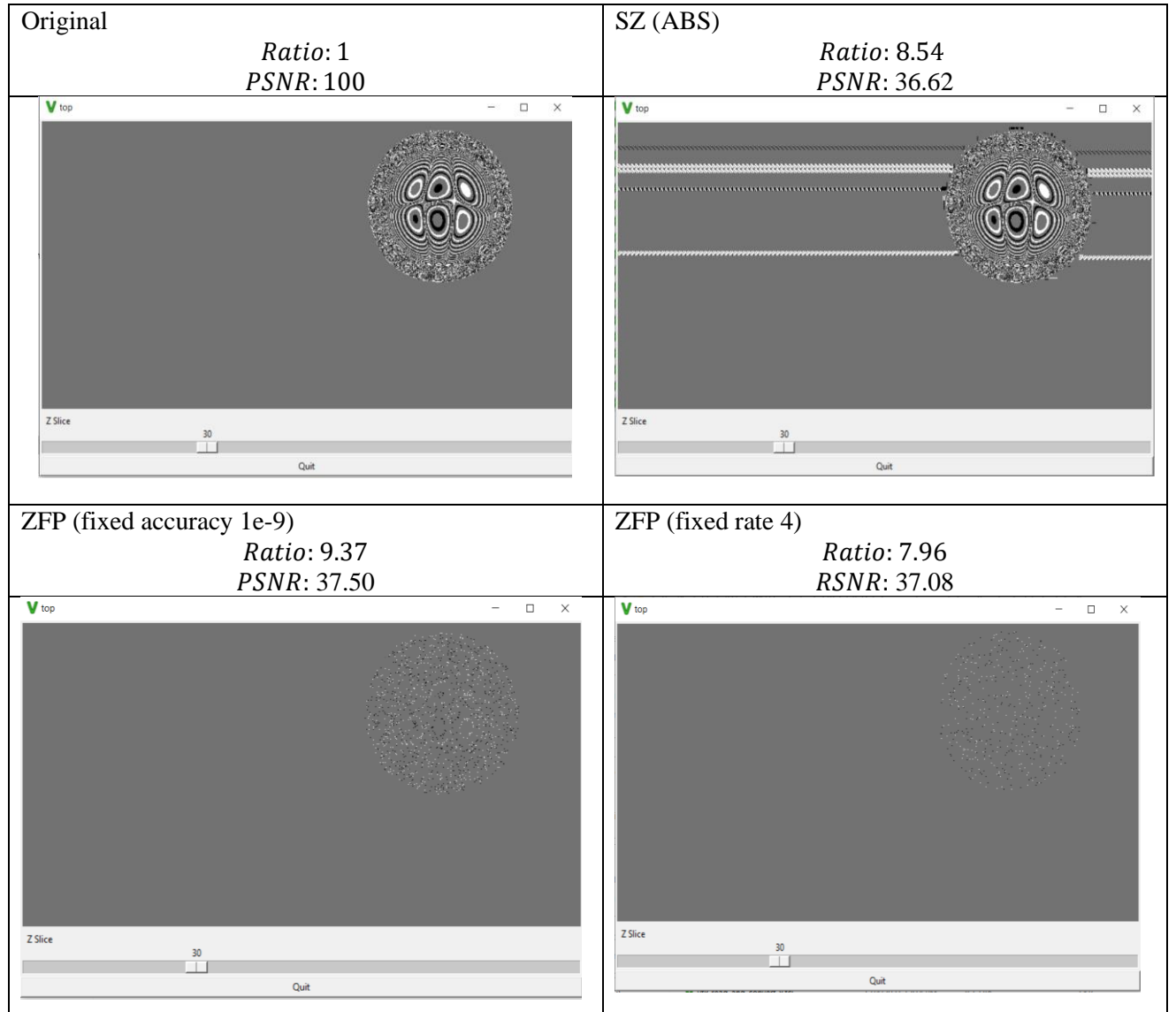


Figure 9. Visualization results on the 001000 file

Figure 9 shows the visualization results on the 001000 file which contains a lot of blank areas. In this case, the compression ratios are all above 7. We chose to discuss this result since this is an inflection point. Since the SZ does not perform well on the blank area, when the compression ratio goes over 8, ZFP performs better based on the PSNR score. However, for the earthquake data, we should focus more on the details rather than the blank area, so we still consider SZ performs better than ZFP in this case.

Overall, SZ with ABS error-bound mode performs better than ZFP with both fixed-accuracy and fixed-rate error-boundaries. Even though the PSNR scores we got from ZFP are higher than the scores we got from the SZ when the ratio is higher than 8 since we are concerned with the details of the earthquake files.

6 Related Work

The impacts that compression makes on visualization have been studied in recent works [7]. uses three different compression algorithms to compress the same data set. Then the compression effect is evaluated by using the set measurement index. This paper focuses on the compression rate and distortion of the compression algorithm [7]. In addition to using compression algorithms, our project also uses visualization tools to visually represent the data set. Not only the performance index is used to compare the advantages and disadvantages of compression algorithms, but also the visual effects of different algorithms to users are compared.

7 Conclusion

In conclusion, we first introduced the dataset information, visualization tools VTK, Tcl script, and two compressors which are SZ and ZFP, and related error-bound modes. During the implementing section, we compressed and decompressed the original data through two compressors with different error-bound modes, visualized the original data and compressed data, and computed the PSNR value between original data and compressed data.

After implementing the study, we concluded that the SZ compressor with the ABS error-bound mode performs better than the ZFP compressor with both fixed-rate and fixed accuracy modes for the earthquake dataset since SZ can keep more detailed information in graphs and we are expected to focus on the details when studying on the earthquake data. However, in the real world, if we are in a situation that is sensitive to blank space, ZFP would perform better since it will not distort blank areas.

For further work, we would like to try more error-bound modes such as value-based error bound, point-wise relative error bound, and peak signal-to-noise ratio error bound and compare the performances among them. Another compressor called T-Threshold could also be tested since it may intuitively perform the best under different circumstances.

Reference

- [1] IEEE Visualization 2006 Design Contest.
[http://sciviscontest.ieeevis.org/2006/data.html#:~:text=TeraShake%202.1%20represents%20a%20physics,the%20Southern%20San%20Andreas%20Fault.&text=TeraShake%202.1%20was%20performed%20by,Center%20\(SDSC\)%20DataStar%20computer](http://sciviscontest.ieeevis.org/2006/data.html#:~:text=TeraShake%202.1%20represents%20a%20physics,the%20Southern%20San%20Andreas%20Fault.&text=TeraShake%202.1%20was%20performed%20by,Center%20(SDSC)%20DataStar%20computer).
- [2] SZ 2.0+: Xin Liang, Sheng Di, Dingwen Tao, Zizhong Chen, Franck Cappello, "Error-Controlled Lossy Compression Optimized for High Compression Ratios of Scientific Datasets", in *IEEE International Conference on Big Data (Bigdata 2018)*, Seattle, WA, USA, 2018.
- [3] SZ 1.4.0-1.4.13: Dingwen Tao, Sheng Di, Franck Cappello, "Significantly Improving Lossy Compression for Scientific Data Sets Based on Multidimensional Prediction and Error-Controlled Quantization", in *IEEE International Parallel and Distributed Processing Symposium (IPDPS 2017)*, Orlando, Florida, USA, 2017.
- [4] SZ 0.1-1.0: Sheng Di, Franck Cappello, "Fast Error-bounded Lossy HPC Data Compression with SZ", in *IEEE International Parallel and Distributed Processing Symposium (IPDPS 2016)*, Chicago, IL, USA, 2016.
- [5] Peter Lindstrom. Fixed-Rate Compressed Floating-Point Arrays. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2674-2683, December 2014. doi:10.1109/TVCG.2014.2346458.
- [6] James Diffenderfer, Alyson Fox, Jeffrey Hittinger, Geoffrey Sanders, Peter Lindstrom. Error Analysis of ZFP Compression for Floating-Point Data. *SIAM Journal on Scientific Computing*, 41(3):A1867-A1898, June 2019. doi:10.1137/18M1168832.
- [7] J. Chen et al., "Understanding Performance-Quality Trade-offs in Scientific Visualization Workflows with Lossy Compression," in *2019 IEEE/ACM 5th International Workshop on Data Analysis and Reduction for Big Scientific Data (DRBSD-5)*, Denver, CO, USA, Nov. 2019, pp. 1–7. doi: 10.1109/DRBSD-549595.2019.00006.
- [8] PSNR Wikipedia, https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio#Definition

Appendix A

Tcl codes for converting binary data file to VTK readable data file:

```
package require vtk
vtkImageReader reader
    reader SetFileName "FileName.bin"
reader SetFileDimensionality 3
reader SetDataScalarTypeToFloat
    reader SetDataByteOrderToBigEndian
    reader SetNumberOfScalarComponents 1
reader SetDataExtent 0 749 0 374 0 99
reader SetDataOrigin 0 0 0
reader SetDataSpacing 800 800 800
vtkImageFlip reslice1
    reslice1 SetInput [reader GetOutput]
    reslice1 SetFilteredAxis 1
vtkStructuredPointsWriter writer
    writer SetInput [reslice1 GetOutput]
    writer SetFileName "FileName.bin.vtk"
    writer SetFileTypeToBinary
    writer Write
exit
```

Appendix B

Tcl codes for visualizing VTK data files:

```
package require vtk
package require vtkinteraction
vtkImageReader reader
    reader SetFileName "FileName.bin.vtk"
    reader SetFileDimensionality 3
    reader SetDataScalarTypeToFloat
    reader SetDataByteOrderToBigEndian
    reader SetNumberOfScalarComponents 1
    reader SetDataExtent 0 749 0 374 0 99
    reader SetDataOrigin 0 0 0
    reader SetDataSpacing 800 800 800
reader Update
    scan [[reader GetOutput] GetWholeExtent] "%d %d %d %d %d %d" \
    xMin xMax yMin yMax zMin zMax
vtkImageFlip reslice1
    reslice1 SetInput [reader GetOutput]
    reslice1 SetFilteredAxis 1
set mag_factor 1
vtkImageMagnify magnify
    magnify SetInput [reader GetOutput]
    magnify SetMagnificationFactors $mag_factor $mag_factor 1
vtkImageViewer viewer2
viewer2 SetInput [magnify GetOutput]
    viewer2 SetZSlice 14
    viewer2 SetColorWindow 2
    viewer2 SetColorLevel 0.1
wm withdraw .
toplevel .top
wm protocol .top WM_DELETE_WINDOW ::vtk::cb_exit
frame .top.f1
set vtkiw [vtkTkImageViewerWidget .top.f1.r1 \
    -width [expr ($xMax - $xMin + 1) * $mag_factor] \
    -height [expr ($yMax - $yMin + 1) * $mag_factor] \
    -iv viewer2]
::vtk::bind_tk_imageviewer_widget $vtkiw
button .top.btn \
    -text Quit \
    -command ::vtk::cb_exit
scale .top.slice \
    -from $zMin \
    -to $zMax \
    -orient horizontal \
    -command SetSlice \
```

```
        -variable slice_number \
        -label "Z Slice"
proc SetSlice {slice} {
    global xMin xMax yMin yMax
    viewer2 SetZSlice $slice
    viewer2 Render
}
pack $vtkiw \
    -side left -anchor n \
    -padx 3 -pady 3 \
    -fill x -expand f
pack .top.f1 \
    -fill both -expand t
pack .top.slice .top.btn \
    -fill x -expand f
tkwait window .
```