JavaSeis

SeisPEG

JPEG-like Seismic Data Compression

Dave Diller, Landmark
Richard Foy, CF Software
Jed Diller, Landmark

Data compression seeks to reduce the number of bits used to store or transmit information. Data compression techniques can be divided into two major families: lossy and lossless. The appropriateness of using these techniques depends on the type of data that is being compressed.

- Lossless decompresses data back to the exact original values.
- Lossy decompresses data back to values that are somewhat different than the original values.

The SeisPEG compression algorithm is a lossy technique, similar to the widely know JPEG standard, but with adaptations for seismic data. For an excellent discussion of the JPEG standard see http://en.wikipedia.org/wiki/JPEG. The essence of JPEG and JPEG-like algorithms such as SeisPEG is that the data is broken into blocks during compression, and lapped transforms are used to avoid blocking artifacts. The advantages of a blocked approach for seismic data include computation efficiency, more effective handling of poorly gained data, and the ability to decompress subsets of the data.

There are three steps in the SeisPEG compression process:

1) Lapped Orthogonal Transform: The purpose of the lapped orthogonal transform is to pack most of the signal energy into a minimum number of coefficients.
2) Quantization: This step reduces the number of unique values that occur in the data in the transformed domain.
3) Minimum redundancy coding: This step represents commonly occurring values in the transformed data with fewer bits than values that do not occur commonly.

The SeisPEG algorithm differs from the JPEG standard in the following respects:

- The SeisPEG algorithm operates on 32-bit floating values instead of small integer values.
- A Lapped Orthogonal Transform (LOT) is used in SeisPEG instead of the Discrete Cosine Transform (DCT).
- The Huffman minimum redundancy coding in SeisPEG uses a pre-computed table that is optimized for seismic data.
- The "zigzag" pattern of run-length encoding in JPEG is not used in SeisPEG, although it may be used in later versions.

The LOT is used in SeisPEG instead of the more commonly used DCT because the LOT better mitigates blocking artifacts and shows greater coding efficiency. There is a family of LOT transforms, and the particular transform that SeisPEG uses was selected by experimentation and demonstrated effectiveness on seismic data. For an in-depth discussion of LOT see the paper entitled "Lapped Orthogonal Transform Coding by Amplitude and Group Partitioning" by Zou and Pearlman, which accompanies this document.

The LOT in and of itself accomplishes no compression. Conversely, it also results in no loss of information; the forward and inverse transforms are completely lossless, other than the slight errors associated with using floating point arithmetic. In this sense it is exactly analogous to the Fourier transforms that we use routinely in seismic processing.

Figure 1. A band-limited dipping event before LOT (left) and after (right).

The LOT in SeisPEG is applied on 2-dimensional frames of data. It would be possible to achieve higher compression ratios if a 3-dimensional LOT was used, but doing so would eliminate the natural fit of SeisPEG into the JavaSeis disk format and would create the need for caching of frames.

Quantization essentially consists of scaling data to smaller values and then rounding to the nearest integer, so that fewer possible discrete values exist in the data, which can therefore be represented with fewer bits. This is exactly what is done for the JavaSeis 8-bit and 16-bit seismic trace formats. Quantization in the LOT domain is performed in exactly the same way as the time domain. This is the only step in SeisPEG compression that introduces error. When quantization is done in the LOT domain (as opposed to the time domain) the resulting round off errors tend to be random in both frequency and time,

which present fewer potential problems in geophysical applications. Because the tolerance to error varies according to how the data will be used, an approximate allowable distortion parameter is specified to adjust the compression ratio versus quality level.

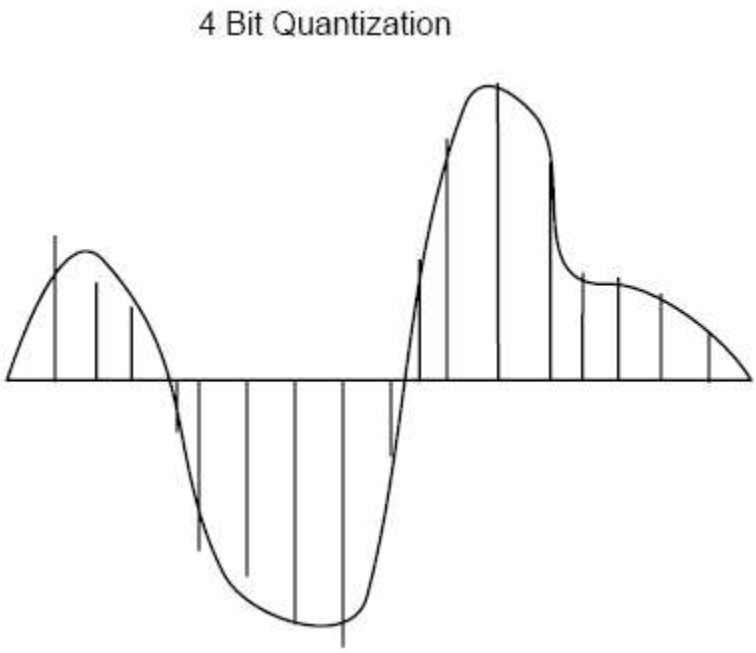

Figure 2. 4-bit quantization of a waveform.

Minimum redundancy coding is the final step in SeisPEG compression. It is used to squeeze out redundancies in the data, which is a perfectly lossless operation. It is particularly effective after a LOT has been applied to the data because the signal tends to be localized by the transform, and after quantization has been applied because there are fewer unique numbers.

There are many minimum redundancy coding techniques in practice, including Huffman coding, run-length encoding, and arithmetic coding. All of these techniques use the underlying principal that frequently occurring values in the data can be represented with fewer bits than infrequently occurring values in the data. This is in contrast to traditional storage of seismic data in which the same number of bits (usually 32) are used to represent every value. SeisPEG uses run-length encoding followed by a specialized Huffman coding algorithm.

The JPEG and SeisPEG algorithms can be adjusted to different quality levels, gaining higher accuracy in exchange for less effective compression. In SeisPEG the approximate amount of distortion to tolerate is specified during the compression process. A value of 0.1 will allow the decompressed dataset to differ from the original dataset by an RMS error of about 10%. Entering a larger value increases the amount of data compression. It is recommended that users test this parameter before compressing and writing seismic data to disk.

Compression Policies

Users are allowed to choose between a **Highest Compression** policy and a **Fastest** compression policy.

Highest Compression compresses the data using the block sizes and transform lengths that produce the highest compression. For highest compression the block size is selected as the largest size that fits the data that is a multiple of 16, up to a limit of 512. Experimentation shows that a block size beyond 512 is substantially slower but with little additional compression. For highest compression the transform length is normally 16. The block size and transform length in time and space are allowed to be different.

The **Fastest** policy compresses the data using the block sizes and transform lengths that are fastest. For fastest compression the block size is the largest size that fits the data that is a multiple of 8, up to a limit of 64. For fastest compression the transform length is 8. Experience shows that the difference in speed between the **Highest Compression** and **Fastest** policies is about 15-20 percent, but the difference in compression ratio is as high as 80 percent. For this reason we recommend the **Highest Compression** policy in most cases.

Trace Headers

Compression of seismic trace headers is accomplished by SeisPEG by transposing each frame of header values and then applying run-length encoding followed by Zip compression (using the java.util.zip package). The run-length encoding algorithm compresses runs of constant value or runs of increasing or decreasing values, which are commonly occurring data patterns in transposed trace headers. The run-length encoding operates on floating point as well as integer format headers. Initial testing on marine seismic data shows a compression ratio of about 12:1 on the trace headers.

Test Results

Initial compression tests using SeisPEG were performed at Landmark in August, 2007 on North Sea marine data. The data in question are well-gained flattened marine CMP gathers.

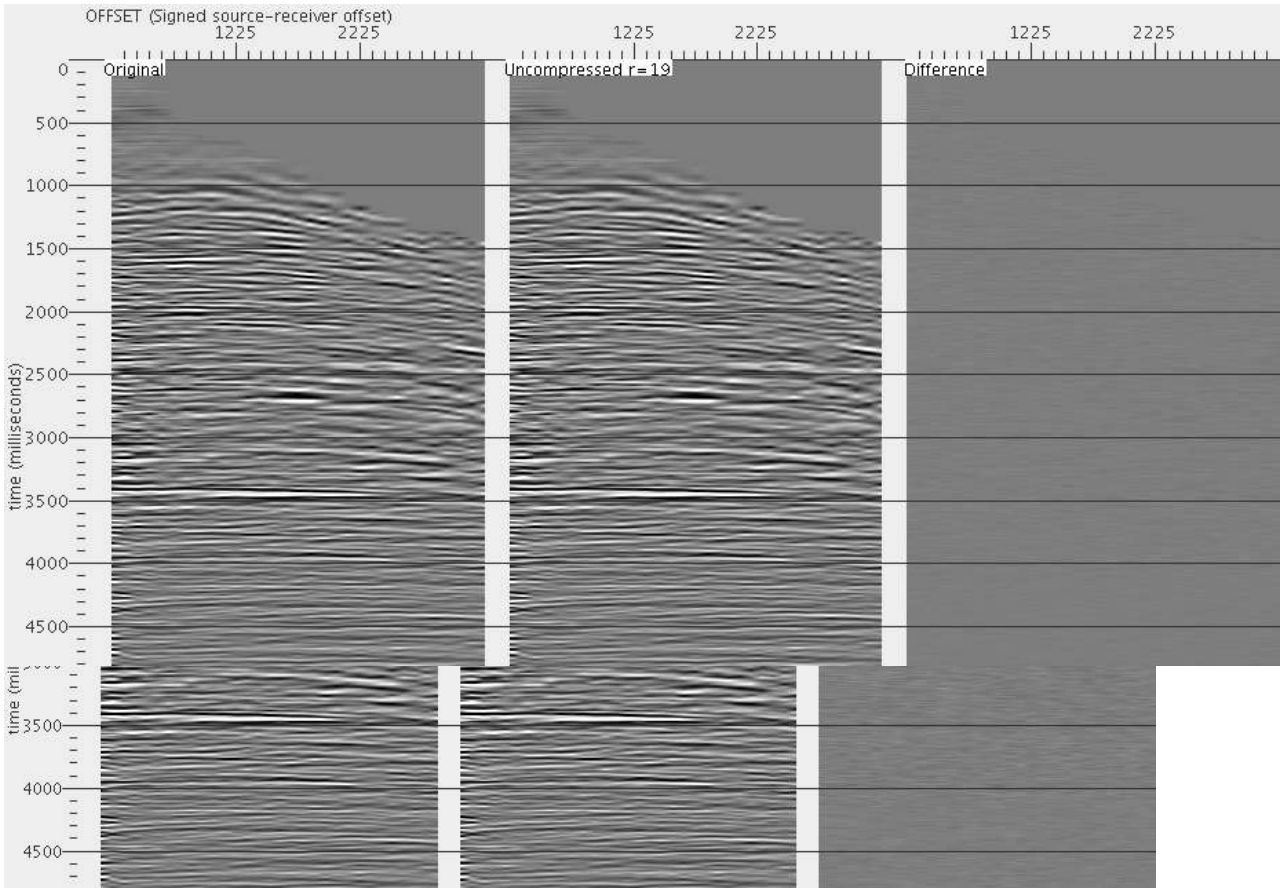

Figure 4. SeisPEG compression of a North Sea marine CMP gather with distortion of 0.1 and a compression ratio of 29:1. Shown are the original data (left), the compress/uncompressed data (center), and the difference (right).

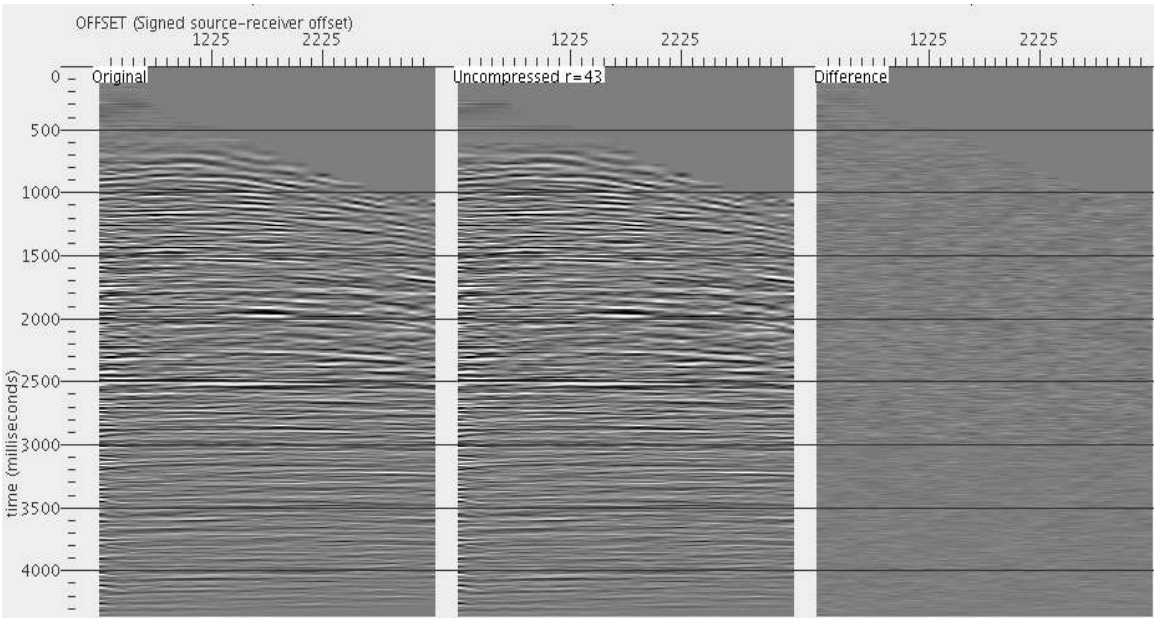

Figure 5. SeisPEG compression of a North Sea marine CMP gather with distortion of 0.2 and a compression ratio of 43:1. Shown are the original data (left), the compress/uncompressed data (center), and the difference (right).

Size vs. Length of Files

JavaSeis is a sparse file format, meaning that all of the space within the files is not necessarily filled with data. However, the "empty" regions do not actually occupy disk space, because all modern file systems are actually tree structures of disk blocks, not contiguous streams of bytes. The file ***length*** of a sparse file, as shown by the Unix ls command, can be much greater than the file ***size,*** as shown by the Unix du command. For example, in the case of the North Sea marine data shown above the compressed trace file length was 117 gigabytes but the actual file size is about 3 gigabytes.

Actual vs. Effective Compression Ratios

Because JavaSeis is a sparse file format and the data for compressed frames are not written contiguously, the amount of disk space that the compressed traces and frames use is controlled by the number of disk blocks that are allocated. The ***actual*** compression ratio is a measure of the number of bytes that are used by the compressed data, but the ***effective*** compression ratio is always less because some portion of the allocated disk blocks are unused. For example, in the case of the North Sea marine data shown above

the actual compression ratio was about 30:1 but the effective compression ratio was about 20:1.

For SeisPEG format we deviate from the Javaseis norm of writing trace samples and trace headers in separate files. The trace sample and trace headers are compressed together and both are written to the trace file (the trace header file will remain 0 length). If trace headers were written in a separate file the result would be an unreasonable waste of disk space because of the block allocation issue for ultra-sparse files.

Threading

SeisPEG uses Java threading to allow concurrent compression/decompression of data. Initial testing shows that the improvement from threading is modest – approximately a 15-30 percent speedup with the addition of each thread. The default number of threads is 2, but it can be reset by setting the runtime Java property "org.javaseis.seiszip.nthreads".

I/O Performance

I/O tests were performed at Landmark in August, 2007 on a single 1.9 gigahertz dual-cpu Linux node attached to a network-attached storage system that has typical maximum I/O rates of approximate 150 megabytes per second. A large seismic dataset was written/read first in JavaSeis 16-bit format and then with SeisPEG using the "fastest" policy and the "highest compression" policy. The difference between these SeisPEG policies is the length of the LOT operator and the block sizes.

	16-bit format	SeisPEG "fastest"	SeisPEG "highest"
Write speed	46 mb/s	36 mb/s	33 mb/s
Read speed	88 mb/s	52 mb/s	46 mb/s

Table 1. Comparison of I/O rates for JavaSeis 16-bit format with SeisPEG "fastest" and "highest compression" policies.

The results of this and other tests indicate that in general the extra computation of the SeisPEG transforms and encoding reduces I/O speeds by about 20-40 percent compared

to that of the JavaSeis 16-bit format. Clearly these I/O rates are fast enough to expect that interactive applications can read and display SeisPEG format at or above the refresh rate of a computer display monitor. Results from I/O tests vary considerably depending on the ratio of compute resource speed to I/O transfer, and depending on the level of compression and the compression policy.

In cases in which the I/O to and from the storage system is saturated we expect that use of SeisPEG format will improve throughput, because the actual data transfer from the storage system to the compute nodes is much lower.

Further Improvements

The current implementation of SeisPEG could benefit from the follow improvements:

- Further work on compression/decompression done in parallel on individual blocks using Java threading. In particular, increasing the granularity of the threading might improve performance.
- Subsets of each frame could be decompressed individually without decompressing the entire frame by exploiting the blocky nature of the algorithm, which would facilitate using the JavaSeis sorting algorithms on SeisPEG datasets.

Known Problems

Gain problems can cause undesirable effects in SeisPEG compression, even though it is a blocky algorithm, because the block sizes are relatively large. For example, compression of raw records with typical decay of amplitude with time may result in the deep data being entirely zeroed out. Very large unedited spikes can also cause problems. After compression the symptom is usually a trace with a large spike having zeroed samples in a zone around the spike.

Compression of seismic data typically has an effect on the data similar to introducing small amounts of white noise, which may somewhat change the effects of subsequent decon. Tests have shown that these effects are small, but you should be aware of them. You may minimize the effects of compression on decon by using options to amortize the operator spectrum on the low and high ends.

Other References

Malvar, H.S., and Staelin, D.H., 1989, The LOT--transform coding without blocking effects: IEEE Transactions on Acoustic, Speech, and Signal Processing, 37, No. 4,553-559.

Malvar, H.S., 1990, Lapped transforms for efficient transform/subband coding: IEEE Transactions on Acoustic, Speech, and Signal Proceeding, 38, No. 6, 969-978.

Pennebaker, W.B., and Mitchell, J.L., 1993, JPEG still image data compression standard: Van Nostrand Reinhold.

Zou, Xiangyu., and William A. Pearlman, 1999, Lapped Orthogonal Transform Coding by Amplitude and Group Partitioning": Applications of Digital Image Processing XXII, Proceedings of SPIE Vol. 3803, 293-304.