Efficient 1-Pass Prediction for Volume Compression

Nils Jensen[1], Gabriele von Voigt[2], Wolfgang Nejdl[1], and Johannes Bernarding[3]

[1] Forschungszentrum L3S, Universität Hannover,
Deutscher Pavillon, Expo Plaza 1,
D-30539 Hannover, Germany
{jensen, nejdl}@l3s.de
[2] Regionales Rechenzentrum für Niedersachsen,
Universität Hannover, Schloßwender Str. 5,
D-30159 Hannover, Germany
vonvoigt@rrzn.uni-hannover.de
[3] Institut für Biometrie und Medizinische Informatik,
Medizinische Fakultät der Otto-von-Guericke Universität Magdeburg,
Leipziger Str. 44, D-39120 Magdeburg, Germany
johannes.bernarding@medizin.uni-magdeburg.de

Abstract. The aim is to compress and decompress structured volume graphics in a lossless way. Lossless compression is necessary when the original scans must be preserved. Algorithms must deliver a fair compression ratio, have low run-time and space complexity, and work numerically robust. We have developed PR0 to meet the goals. PR0 traces runs of voxels in 3D and compensates for noise in the least significant bits by way of using differential pulse-code modulation (DPCM). PR0 reduces data to 46% of the original size at best, and 54% on average. A combination of PR0 and Worst-Zip (Zip with weakest compression enabled) gives reductions of 34% at best, and 45% on average. The combination takes the same or less time than Best-Zip, and gives 13%, respectively 5%, better results. To conduct the tests, we have written a non-optimised, sequential prototype of PR0, processed CT and MRI scans of different size and content, and measured speed and compression ratio.

1 Introduction

Volume compression means to reduce the mean number of bits to describe regularly positioned scalars (voxels) in 3D. Volume decompression reconstructs the data. Information that is meaningful to the user must be preserved.

The paper specifies novel algorithms, PR0, to compress and decompress volumes in a lossless way. The advantage of a lossless algorithm is it can be combined with lossless and lossy schemes to optimise performance, accuracy of data, or compression ratio. But to design an efficient solution that meets the goals is challenging. Compvox [3] uses seven passes, compared to one in PR0.

PR0 uses predictors [10] in raster-scan order on each scalar, slice by slice. It uses seven predictors to estimate values in a slice by way of referring to the

H. Kalviainen et al. (Eds.): SCIA 2005, LNCS 3540, pp. 302–311, 2005.

previously scanned values. At any time, only two slices must be kept in main memory. Dependent on which predictor triggers for a local data pattern (the first-order continuation along the 12-neighbourhood), PR0 appends the static Huffman code of the predictor and optionally writes the difference between the predicted and the correct value in a fixed number of bits. One more predictor can always match, because it does not refer to previously scanned data but uses a separately maintained dictionary of uncompressed scalar values. Decompression works in the same way. The challenge was to design predictors that would match often. A novel approach is to collect some predictors under one binary code and to define a mechanism that helps the decompressor to disambiguate between them. The paper specifies the mechanism.

Areas of application are medical and scientific data archiving, multimedia streaming, and 3D-texture compression on graphics cards.

Section 2 specifies related work in lossless volume compression. Section 3 specifies PR0. The Sections 4 and 5 give interpreted results from test runs. The concluding Section draws on future work.

2 Lossless Volume Compression

2.1 Rationale for the Development of Specialised Algorithms

Zip is the most widely used compression and decompression tool. Implementations of Zip follow the LZW algorithm [12][10]. LZW's advantages are availability, flexibility, speed, maturity, and efficacy. But for volume compression, LZW or any other general-purpose algorithm is sub-optimal because it must "rediscover" information that is intrinsic to the generic structure of volumes that are inspected by human beings. To use LZW to compress volumes uses more processor (CPU) cycles and memory than necessary.

Further, volumes are more and more frequently used, and to manage them in an economic way is important for the success of many areas of application. Examples are archives that host Terabytes of medical data on discs, Web-based components that transfer Gigabits of multimedia data over the Internet, and graphic cards that optimise memory use and bus traffic.

The paper discusses a promising approach toward real-time, lossless processing of volumes. We do not discuss lossy compression [8] to give the user control over the data quality degradation process, by way of selecting and composing data removal and information removal algorithms that complement each other. This means to separate quantisation from compression.

2.2 Specialised Lossless Algorithms

We survey prior work about volume compression before we specify our new approach in Section 3. Related applications have been discussed [6].

Fowler and Yagel [3] specify the Compvox algorithm that traces runs of voxels in 3D and uses DPCM to achieve compression ratios of 2:1 in the presence of noise in the input data. Compvox uses one predictor that scans orthogonal neighbours

of a voxel to estimate it and uses a combination of delta and Huffman encoding to store estimated values. The algorithm determines the weighting co-efficients to prepare the calculations in seven passes, each time over all input data.

Researchers have experimented with techniques that are known from 2D image processing [4]. They achieve compression ratios of 3:1 by means of combining background suppression and intra-slice prediction. The authors report weaker results from the application of, surprisingly, inter-slice prediction, multi-resolution pyramids, symmetry coding for which they match two halfs of a volume to eliminate redundant data, and model-based coding for which they match a volume with a reference volume and encode only the difference between them. The compression ratio is only representative for volumes of MRI brain scans, the authors neither report results from compressing other volume types nor execution times.

A conceptually simple method for the lossless compression of volumes has been proposed [7]. The input data are decomposed in subbands and block-sorted. An arithmetic encoder post-processes data. The advantage of the approach is that the algorithm supports progressive decoding. Part of the method is equivalent to background suppression [4] because it classifies the input data in relevant and irrelevant regions. The reported reduction of 3:1 is verified for one dataset.

Steingötter and colleagues [11] compare how run-length-encoding (RLE), Huffman encoding, and JPEG, for which each is enhanced to find inter-slice correlation, compress a medical scan of the human thorax. They claim that compression ratios of 3:1 can be achieved but they do not report details.

Other systems use Wavelets, most predominant are 3D-SPIHT and its successor, AT-SPIHT [1]. SPIHT means to partition input data in a hierarchical tree to sort, compress, and decompress data in a lossless way. SPIHT stores the Wavelet-coefficients that are created during the computation in bit-plane order, orders them by magnitude, and extracts the least significant bits. For each category of importance to the 3D image, the data are written to the output stream. Enhancements to improve the efficiency of the algorithm are regularly reported, in this case AT-SPIHT. It unbalances the Wavelet-coefficient-tree to achieve better compression ratios than 3D-SPIHT. The reported reduction of 25% of the original size of a volume is the best result which has been reported. Another advantage is that SPIHT supports progressive decoding, as every Wavelet-based system. The disadvantages are that SPIHT uses more than one pass to remove data, and that the algorithm is complex, which restricts its implementation in hardware. Practically relevant is SPIHT is patented.

3 The PR0-Algorithm

3.1 Overview

The idea is to separate the topology from the topography of the volume and compress each by way of using a different algorithm. Further, the algorithm must work for any kind of volumes in an effective way where large connected regions of a volume have voxels of similar values. We can then define that the

connectivity is the topology, and that the topography is the represented set of scalar voxel values.

PR0 splits in one pass a volume in a dictionary of uncompressed values and a sequence of selected predictor codes. They interleave, if necessary, with DPCM bits to make predictor matching robust against noise in the least significant bits.

3.2 Design

We have selected the following 8 predictors in empirical test runs to compress and decompress a volume to support training (bonsai). q is a queue that contains uncompressed values, p is a predictor, and v is a voxel value at a 3D-position.

$$p_1(x,y,z) = (v(x-1,y,z-1) + v(x+1,y,z-1))/2 \quad (1)$$
$$p_2(x,y,z) = (v(x,y-1,z-1) + v(x,y+1,z-1))/2 \quad (2)$$
$$p_3(x,y,z) = (v(x-1,y-1,z-1) + v(x+1,y+1,z-1))/2 \quad (3)$$
$$p_4(x,y,z) = (v(x-1,y+1,z-1) + v(x+1,y-1,z-1))/2 \quad (4)$$
$$p_5(x,y,z) = (v(x-1,y,z) + v(x,y-1,z))/2 \quad (5)$$
$$p_6(x,y,z) = v(x,y,z-1) \quad (6)$$
$$p_7(x,y,z) = (v(x-1,y,z) + v(x,y-1,z) + v(x-1,y-1,z))/3 \quad (7)$$
$$p_8(x,y,z) = \texttt{fifopop(q)} \quad (8)$$

The algorithm probes for each voxel all predictors in sequence if they reconstruct the original value properly. It encodes the first one that matches and optionally modifies it to store DPCM bits. The C-style pseudo code for the compressor is (t is a heuristic function which is explained below)

```
for all v in input data, s for output stream, q for queue
  for all i in p_i from predictors and t_i from triggers
    if | p_i(x, y, z) - v(x, y, z) | <= epsilon and t_i(x, y, z) == 1
    then fifopush(s, c) where c is the partial code of predictor id i
    if i == 8 then fifopush(q, v(x, y, z))
    delta = (p_i(x, y, z) - v(x, y, z))
    fifopush(s, tocode(delta))
```

The pseudo code for the decompressor is similar:

```
read queue q
for all c in input stream s, v for output data
  for all i's in the set C that is associated with the partial code c,
    p_i are predictors, t_i are triggers
    if i == 8 then v(x, y, z) = fifopop(q)
    else if t_i(x, y, z) == 1 then v(x, y, z) = p_i(x, y, z)
      if c contains modulation code delta then
        v(x, y, z) += fromcode(delta)
```

To use t reduces the number of bits to specify which predictor was selected, by way of using implicit information in the input data. We call the principle *partial*

predictor encoding, which works in detail as follows: some predictors (in PR0, the first six) share the same predictor code and the decompressor must disambiguate between the sub-predictors if it encounters a code that refers to one of them. The compressor and decompressor use, in this case, the heuristic t to resolve the conflict by means of indicating which predictor would be the most suitable for the current data pattern. In PR0, we assign each of the first five predictors a trigger function to decide which of the sub-predictors matched (ϵ is an arbitrary but fixed threshold that bounds the maximum error).

$$t_1(x, y, z) = (\epsilon \geq |v(x-1, y, z-1) - v(x+1, y, z-1)|) \tag{9}$$
$$t_2(x, y, z) = (\epsilon \geq |v(x, y-1, z-1) - v(x, y+1, z-1)|) \tag{10}$$
$$t_3(x, y, z) = (\epsilon \geq |v(x-1, y-1, z-1) - v(x+1, y+1, z-1)|) \tag{11}$$
$$t_4(x, y, z) = (\epsilon \geq |v(x-1, y+1, z-1) - v(x+1, y-1, z-1)|) \tag{12}$$
$$t_5(x, y, z) = (\epsilon \geq |v(x-1, y, z) - v(x, y-1, z)|) \tag{13}$$

The seventh and the eighth predictor are explicitly encoded in the compressed data and therefore, have no trigger functions associated with them.

The eighth predictor is always applicable because it stores a complete value in the queue q for decompression. The number of entries in the queue matches the number of times the default predictor was encoded.

3.3 Code Format

Because we use trigger functions, we approximately halve the number of bits to identify the predictors and add a 1-bit DPCM-code. This uses at most 3 bits per predicted value. The compressor attaches more DPCM-bits when it processes volumes that comprise more than 8 bits per voxel because then the noise spreads over more than one bit. We have achieved a good compression ratio with 6 bits to encode the complete modulation. Table 1 specifies the codes to identify selected predictors to estimate 8-bit data types (six predictors share a code).

Table 1. Huffman codes (with prefix-condition)

Predictor id i	Perfect match	Correction: -1	Correction: +1
1..6	01	001	000
7	110	101	100
8	111	–	–

4 Evaluation

4.1 Setup

The first author implemented PR0 in C and compiled the parts `PREDICT_INCORE` and `PREDICT_FAST`. `PREDICT_SHORTDICT` in combination with the documented

Table 2. Samples (see Fig 1 from left to right, top to bottom), size is given in voxels

Name	X-size	Y-size	Z-size	Bits per voxel	Number of volumes
bonsai	256	256	128	8	1
brain	256	256	109	16	1
engine	256	256	128	8	1
head	256	256	113	16	1
crt-2	256	256	192	16	1
funcbrain	64	64	32	16	900

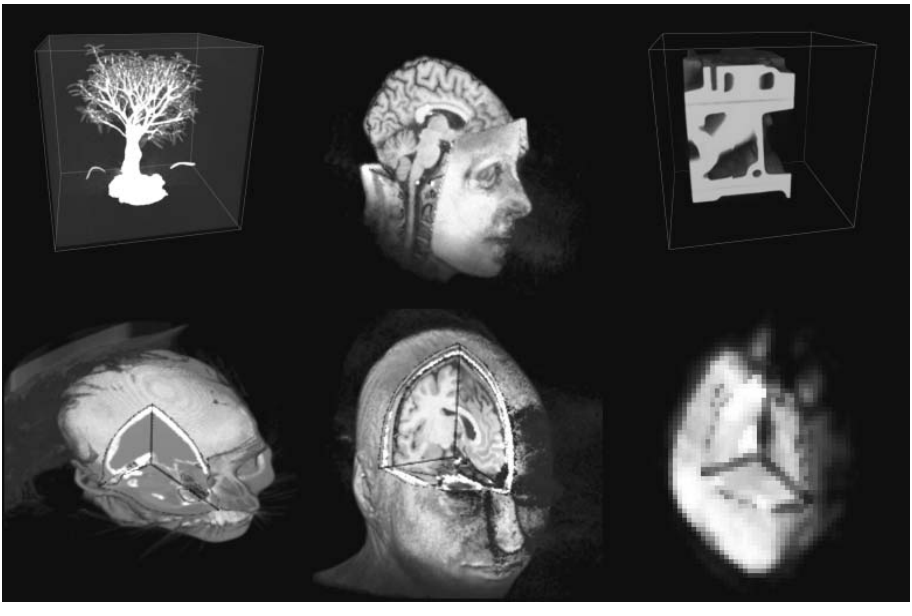

Fig. 1. The volumes for the evaluation, visualized by OpenQVis and MRIcro

typedefs enabled support for 16-bit voxels. The executable was optimised for speed on a Pentium 4. The source code did not contain optimisations. The author used MS Windows XP and Microsoft C++ Version 13.10.3077. His workstation was a Dell Precision 340 with 37 GB harddisc, 512 MB main memory, and a Pentium 4 at a clock rate of 2.4 GHz.

The first author took samples from [2][9] and converted them to Little-Endian format and merged them to one file each. The fourth author provided a 24.5 MB CRT scan (crt-2) and a sequence of volumes (funcbrain) that describe in 225 MB functional measurements of brain activity over time. Table 2 specifies the datasets.

4.2 Performance

The first author compressed each file once in mode −1 (Worst-Zip) and in mode −9 (Best-Zip) of Info-Zip version 2.3 under Cygwin 1.3, on top of the XP installation. He used Cygwin for the evaluation of Zip's performance because it was already available in the distribution and usable with negligible overheads. The author used PR0 for each file and compressed the .pr0 files with Info-Zip once in mode −1 and once in mode −9 (the latter was omitted in the Tables because it is two to three times slower and saves just a few KB). He measured how long each action took for each dataset. The complete process was repeated twice, except for funcbrain, and the results averaged. To retrieve fine measurements, the author enabled `PREDICT_BENCHMARK` in the PR0 implementation to use `GetTickCount()` of the MS Windows API to calculate and print the average time for processing each slice of the crt-2 file. The fine measurements were done in a separate test run. We then compiled the results.

4.3 Results

PR0 reported less than 8 milliseconds to process $256^2 \cdot 16$ bit. To compress crt-2 took 896 ms. Table 3 gives the compression in the percentage of the size of the original uncompressed file. Table 4 specifies execution time. (The execution times to process funcbrain were rounded to seconds.)

Table 3. Comparison by the size of the output data, given by the percentage of the size of the input data in bytes

Name	PR0	Worst-Zip	Best-Zip	Worst-Zip, PR0 pre-processes
bonsai	58	50	47	50
brain	58	57	55	53
engine	46	40	38	34
head	52	51	50	44
crt-2	56	49	48	45
average	54	49	48	45
funcbrain	59	49	46	46

Table 4. Comparison by the execution time in sec, for compression / decompression

Name	PR0	Worst-Zip	Best-Zip	Worst-Zip, PR0 pre-processes
bonsai	1.8 / 1.5	0.3 / 1.0	3.7 / 1.0	0.7 / 0.3
brain	1.7 / 2.0	2.3 / 1.0	5.0 / 1.0	1.7 / 1.3
engine	1.5 / 1.2	1.0 / 0.7	3.7 / 1.0	1.0 / < 0.1
head	1.7 / 2.0	2.3 / 0.7	4.0 / 1.0	1.3 / 1.0
crt-2	2.9 / 3.7	2.0 / 3.7	19.7 / 2.3	4.0 / 1.7
average	1.9 / 2.1	1.6 / 1.4	7.2 / 1.3	1.7 / 0.9
funcbrain	49.0 / 43.0	39.0 / 53.0	215.0 / 42.0	30.0 / 22.0

5 Discussion

5.1 Interpretation of Results

Tables 3 and 4 specify that PR0 performs well in comparison with the standard tool. PR0 gives nearly the same compression ratio as Zip. In both PR0 and Zip, the decompression time is proportional to the target file size, therefore the better the compression, the faster the decompression is. This is shown by Best-Zip which generates an excellent compression ratio in general, takes long to compress, and takes short time to decompress. Best-Zip's poor compression speed for the crt-2 and funcbrain datasets is probably due to a non-optimal access scheme encountered in the dictionary management and the inability to recognise the 16-bit format of the voxel stream and the sequences of slices of static extent, which PR0 handles by definition in an optimal way.

To zip PR0's output reduces it further. The Tables 3 and 4 show it. We can produce the most remarkable reduction rate with the engine dataset, because it describes a simple structure. Generally, a combination of Worst-Zip and PR0 generates better compression rates than Best-Zip alone, with the exception of the bonsai volume (3% loss of compression) and the funcbrain sequence of volumes (no improvement of compression but drastic reduction of execution time). A better compression for larger volumes indicates that the latter is caused by the implementation of PR0 which does not compress volume boundaries. A modified implementation will avoid the problem (see Section 5.3).

Without closer inspection, one could assume that PR0 gives nice but not dramatically better results. A combination of PR0 and Worst-Zip could not replace Best-Zip because the sum of the execution times of PR0 and Worst-Zip is greater than those for Worst-Zip and not substantially better than Best-Zip for some cases. And for some applications, the loss of decompression performance could be a disadvantage. But none of these arguments is backed up by the measurements. The times to execute the algorithm, which are shown in the Tables, include measurements for the disc access routines, which dominate compression and decompression as one sees from comparing the table entry for PR0 and ct-2 with the detailed measurement: compression takes 0.9 sec and read-write file access more than twice the time, that means read, (de-)compress, and write take equally long. Two disc accesses can be removed when combining PR0 and Zip by way of forwarding intermediary data streams in main memory. Hence, a combination of PR0 and Zip can minimise the file size *and* execution time. Our data indicate that this effect will amplify in favour of PR0 when the size of the volumes increases. Each discussed improvement is small but accumulates to substantial savings when it is applied to support batch- or streaming-applications.

In summary, the evaluation of PR0 has indicated advantages of our algorithm:

1. Zip almost consistently compresses data better when they are pre-processed with PR0.
2. To combine Zip at compression mode -1 (weakest) with PR0 takes substantially less time as Zip at compression mode -9 (strongest). This is most evident for the examples crt-2 and funcbrain.

The disadvantage is that the current implementation of PR0 is less efficient than Worst-Zip, which surprises us because PR0 should theoretically be faster due to the simpler management of the data dictionary. Hence, we believe an optimal implementation of PR0 would at least match Worst-Zip's runtime behaviour.

5.2 Comparison with Prior Work

PR0 has a fair compression ratio for realistic data, is small (about 300 lines of code), uses simple predictors that are selected during one compression run, and is numerically robust because only few fundamental operations are necessary to compress and decompress data. Because the algorithms are lossless, they do not generate hidden errors by accident.

The algorithms can be parallelised because the referenced dataset is local and it exploits the multi-dimensional data coherence and constant constraints, for example the size of the input data slices.

Last, one achieves fine-grained balancing between cost/profit parameters by way of replacing or removing predictors.

Code compactness, greater potential for parallelisation schemes, and simplicity as well as customisability are factors that have not been addressed elsewhere to our knowledge. Instead, reports of similar approaches often focus on the data reduction as the single criterion. However, more factors are relevant [5][10].

In specific comparisons, Fowler and Yagel's [3] implementation compresses approximately equally to a combination of PR0 and Zip and is three times slower than Zip. Different to [4] which achieve better average compression ratios, PR0 is useful for a larger class of volumes, not just medical datasets. The approach by [7] achieves a better reduction of data than PR0, but as the authors admit, has weak compression performance. Finally, AT-SPIHT [1] does not meet fixed real-time constraints but PR0 does.

5.3 Limitations

(i) We did not specify the entropy of the volumes. (ii) PR0's implementation does not compress boundaries. Predictors could read constants to compensate nil values. (iii) DPCM bits are uncompressed. They could be Huffman-encoded.

6 Conclusions

We have specified novel predictor-based compression and decompression algorithms. The design follows recommendations to balance between compression ratio and speed [10]. As demonstrated, the algorithms combine well with other codecs. The source code and data material are at www.l3s.de/~jensen/lossless/.

Researchers should investigate the quality of the selected predictors in a theoretical analysis and improve them. Developers can combine PR0 and other codecs, evaluate the benefit of adaptively selecting predictors, and optimise the reference implementation. We speculate that an FPGA-implementation of PR0 will process animation sequences of volumes in real-time.

Acknowledgments

We thank S. Olbrich for comments, and the DFG for funding the project EVITA.

References

1. Cho, S., Kim, D., Pearlman, W.A.: Lossless Compression of Volumetric Medical Images with Improved Three-Dimensional SPIHT Algorithm. In: Honeyman-Buck, J.C. (ed.): Journal of Digital Imaging. Vol. 17. **1** (2004) 57–63
2. Engel, K.: Pre-integrated Volume Rendering. At: http://wwwvis.informatik.uni-stuttgart.de/~engel/{Bonsai.zip, Engine.zip}. (2001) Accessed: 28–Jan–05
3. Fowler, J.E., Yagel, R.: Lossless Compression of Volume Data. In: Kaufman, A., Krüger, W. (eds.): Proceedings of the 1994 Symposium on Volume Visualization. ACM Press, New York (1994) 43–50
4. Hargreaves, B., Johanson, B., Nayak, K.: Lossless Compression of 3D MRI Brain Images. At: http://ise.stanford.edu/class/ee392c/demos/hargreaves_johanson_nayak/. (1997) Accessed: 2–Dec–04
5. Herrero, I., Salmeron, J.L.: Using the DEA Methodology to Rank Software Technical Efficiency. In: Crawford, D. (ed.): Comm. of the ACM. Vol. 48. **1** (2005) 101–105
6. Jones, M.: Distance Field Compression. In: Skala, V. (ed.): Journal of WSCG. Vol. 1–3. (2004) 199–204
7. Klappenecker, A., May, F., Beth, Th.: Lossless Compression of 3D MRI and CT Data. In: Laine, A.F., Unser, M.A., Aldroubi, A.(eds.): Proceedings of SPIE on Wavelet Applications in Signal and Imaging Processing VI. (1998) 140–149
8. Krishnan, K., Marcellin, M., Bilgin, A., Nadar, M.: Compression / Decompression Strategies for Large Volume Medical Imagery. In: Ratib, O.M., Huang, H.K. (eds.): Proceedings of SPIE Medical Imaging 2004: PACS and Imaging Informatics. Vol. 5371. (2004) 152–159
9. Levoy, M.: The Stanford Volume Data Archive. At: http://graphics.stanford.edu/-data/voldata/{CThead.tar.gz, MRbrain.tar.gz}. (2004) Accessed: 28–Jan–05
10. Salomon, D.: Data Compression – The Complete Reference. 3rd edn. Springer-Verlag, Berlin Heidelberg New York (2004)
11. Steingötter, A., Werner, C., Sachse, F., Dössel, O.: Kompression drei- und vierdimensionaler medizinischer Bilddaten. In: Biomedizinische Technik. Vol. 43. (1998) 478–479
12. Welch, T.A.: A Technique for High-Performance Data Compression. In: You, S.S. (ed.): IEEE Computer. Vol. 17. **6** (1984) 8–19