Accelerating the TigerVNC Encoder
(For Fun and Profit)

Version 1d, 3/9/2012 -- The VirtualGL Project

---

1 Introduction

The report "From Tight to Turbo and Back Again: Designing a Better Encoding Method for TurboVNC" revisits a study conducted in 2008, during which a procedure was established for analyzing the performance of various VNC encoding methods at the low level. The goal of this study was to improve the compression efficiency of TurboVNC without sacrificing any of its performance. The procedure involved benchmarking a particular VNC encoder using RFB protocol captures from 20 different workloads, both 2D and 3D. In that report, the almost pure high-speed JPEG encoder from TurboVNC 0.4 served as a performance target, whereas TightVNC 1.3.9 served as a compression efficiency target. Ultimately, an encoding method was designed which, for the datasets of interest, performed as good as or better than TurboVNC 0.4 in all cases and approached the compression ratio of TightVNC 1.3.9 in many cases.

Since the introduction of TigerVNC in 2009, there has been a desire to perform the same analysis on the TigerVNC encoder. It has been known since that time that TigerVNC's high-level performance was significantly slower than that of TurboVNC, but the reasons behind this were unknown, since the two solutions shared the same high-speed JPEG codec (libjpeg-turbo.)

2 Tools and Methodology

The tools, equipment, and methodology used in this report were identical to those used in the afore-mentioned "Tight to Turbo" report, which is included by reference. However, since the amount of data obtained for this report was significantly greater than the amount of data obtained for the previous report, the data in this report is discussed in relative rather than absolute terms.

The goal of this research was to produce a series of modes in TigerVNC which duplicated both the CPU time and compression ratio of the "Tight + Perceptually Lossless JPEG", "Tight + Medium Quality JPEG", "Lossless Tight", and "Lossless Tight + Zlib" modes in TurboVNC, all without compromising the levels of performance that users already enjoyed with TigerVNC 1.1. A secondary goal was to explore the usefulness of the various compression levels in TigerVNC.

If, for any particular dataset, a change to a particular mode produced worse compression efficiency or throughput than the same mode in TigerVNC 1.1, a method for restoring the performance of TigerVNC 1.1 for that dataset and mode is discussed.

2.1 Metrics

Two metrics are used in this report: "Relative Compression Ratio" and "Speedup":

$$\text{Relative Compression Ratio} = \frac{\text{Compression Ratio}_{New}}{\text{Compression Ratio}_{Baseline}}$$

$$Speedup = \frac{\text{CPU Time}_{Baseline}}{\text{CPU Time}_{New}}$$

"Speedup" is the same as "relative throughput". In both cases, a value of 1.0 indicates that the new version was equivalent to the baseline. Values < 1.0 indicate that the new version was worse than the baseline, and values > 1.0 indicate that the new version was better than the baseline.

3 Results

3.1 Perceptually Lossless JPEG

To establish a baseline, the TurboVNC encoder was benchmarked using the `compare-encodings` benchmark with the canonical set of 20 RFB session captures. The settings for "Tight + Perceptually Lossless JPEG" were used, except that the JPEG quality was dialed down to 92 in order to match the JPEG Quality Level 8 setting in TigerVNC. Next, the unmodified TigerVNC 1.1 encoder was benchmarked using JPEG Quality Level 8 and Compression Levels 1, 2, 3, 6, and 9.

TigerVNC 1.1 vs. TurboVNC: 4:4:4 Subsampling, JPEG Quality = 92

Dataset	Relative Compression Ratio (vs. TurboVNC Baseline)	Speedup (vs. TurboVNC Baseline)
TigerVNC 1.1 (Compression Level 1)	Min: 0.505 Avg: 0.903 Max: 1.91	Min: 0.417 Avg: 0.709 Max: 0.911
TigerVNC 1.1 (Compression Level 2)	Min: 0.696 Avg: 1.05 Max: 1.84	Min: 0.398 Avg: 0.732 Max: 0.937
TigerVNC 1.1 (Compression Level 3)	Min: 0.847 Avg: 1.15 Max: 1.85	Min: 0.366 Avg: 0.652 Max: 0.886

Dataset	Relative Compression Ratio (vs. TurboVNC Baseline)	Speedup (vs. TurboVNC Baseline)
TigerVNC 1.1 (Compression Level 6)	Min: 1.02 Avg: 1.20 Max: 1.97	Min: 0.244 Avg: 0.417 Max: 0.853
TigerVNC 1.1 (Compression Level 9)	Min: 0.965 Avg: 1.16 Max: 1.96	Min: 0.0320 Avg: 0.0949 Max: 0.676

The default compression level in TigerVNC 1.1 (level 6) uses a relatively high amount of Zlib compression, and this proved to be one of the factors in TigerVNC 1.1's lackluster performance relative to TurboVNC. Reducing the compression level improved the situation, but still the best relative performance that could be achieved was with Compression Level 2. Even that mode compressed as much as 30% less efficiently and was as much as 60% slower than TurboVNC.

Also note that, in TigerVNC 1.1, compression levels higher than 6 were utterly useless in conjunction with perceptually lossless JPEG. Switching from Compression Level 6 to Compression Level 9 actually reduced the compression ratio of almost all of the datasets, and it increased the average CPU time by a factor of 4.4. The jump from Compression Level 3 to Compression Level 6 was of only marginal usefulness, increasing the compression ratio by an average of only 4.3% in exchange for an average 56% increase in CPU time.

The first step in diagnosing the performance disparity between TurboVNC and TigerVNC was to examine the low-level behavior of the two encoders. Four major differences were discovered:

1. If the connected viewer supports “last rectangle encoding” (a TightVNC extension that allows the server to send an arbitrary number of rectangles in a framebuffer update), then TurboVNC and TightVNC 1.3.x will attempt to identify areas of solid color in a rectangle prior to dividing it into subrectangles. Thus, the areas of solid color will dictate how the rectangle is divided (hence the need for last rectangle encoding, since it's impossible to predict how many subrectangles will be generated without actually performing the computation.) TigerVNC did not support last rectangle encoding, and thus it divided the rectangle into subrectangles first, then it determined whether each subrectangle was solid or not. This could lead to situations in which large areas of solid color are sent using indexed color or JPEG encoding, simply because there are tiny areas of high color depth at their edges. It was suspected that this was one of the factors limiting the performance of TigerVNC 1.1.

2. As with TightVNC, TigerVNC computed the palette threshold based on the size of the subrectangle, but unlike TightVNC, TigerVNC would then clamp the threshold to a value of 96 if JPEG was enabled. TurboVNC instead sets the palette threshold to a static value (24) regardless of the subrectangle size. It was suspected that this difference in behavior might have been causing TigerVNC's mix of JPEG and indexed color subrectangles to be weighted too heavily in favor of indexed color.

3. As with TightVNC, the maximum subrectangle size and width that TigerVNC would encode varied based on the compression level. It was suspected that the use of small subrectangles was

increasing the encoding overhead for the lower compression levels.

4. TigerVNC did not send JPEG subrectangles if the subrectangle size was less than 1024 pixels or if either of the subrectangle dimensions were less than 8. This was suspected of throwing off the balance between JPEG and indexed color subrectangles, since some subrectangles can be long and narrow but still have enough pixels to compress efficiently using JPEG.

Rather than spend a tremendous amount of time benchmarking every possible permutation of the above, it was decided instead to attempt to exactly duplicate the output of the TurboVNC encoder and see where it stood relative to TigerVNC 1.1. This involved:

- Porting the solid subrectangle pre-computation code from the TurboVNC encoder to the TigerVNC encoder (this required implementing last rectangle encoding as well)

- Tweaking the TigerVNC encoder such that it used a static rather than a variable palette threshold whenever JPEG encoding was enabled

- Tweaking the TigerVNC encoder such that it used a maximum subrectangle size of 65,536 and a maximum subrectangle width of 2048 for all compression levels

- Tweaking the TigerVNC encoder such that it did not limit the size of subrectangles that were candidates for JPEG encoding

- A great deal of debugging

Once it was verified that the precise subrectangle mix from TurboVNC could be duplicated for all datasets, the benchmarks were re-run using the new encoder. The new numbers revealed that some datasets were still falling short of the TurboVNC performance target. One reason for this was discovered to be the fact that, when encoding a solid subrectangle, the TigerVNC encoder would perform pixel translation on all of the pixels in the subrectangle, even though only the first one was relevant. This became a more visible phenomenon because the new encoder was sending solid subrectangles much more frequently than the old encoder did. A second reason for the performance lag was that the JPEG encoder was using too small of a buffer and was adding an extra, unneeded buffer copy. The JPEG encoder was moved into a separate class and re-designed so that it avoids the extra buffer copy, and the encoding buffer was increased in size to 128k to avoid the overhead of buffer re-allocation.

The resulting encoder, hereafter referred to as “TigerVNC 1.2”, was then compared to the TurboVNC and TigerVNC 1.1 encoders:

TigerVNC 1.2 vs. TurboVNC and TigerVNC 1.2 vs. TigerVNC 1.1: 4:4:4 Subsampling, JPEG Quality = 92

Dataset	Relative Compression Ratio (vs. TurboVNC Baseline)	Speedup (vs. TurboVNC Baseline)	Relative Compression Ratio (vs. Same Compression Level in TigerVNC 1.1)	Speedup (vs. Same Compression Level in TigerVNC 1.1)
TigerVNC 1.2 (Compression Level 1)	Min: 0.989 Avg: 1.00 Max: 1.02	Min: 0.968 Avg: 1.06 Max: 1.16	Min: 0.525 Avg: 1.11 Max: 1.98	Min: 1.22 Avg: 1.49 Max: 2.33
TigerVNC 1.2 (Compression Level 2)	Min: 1.00 Avg: 1.17 Max: 1.84	Min: 0.707 Avg: 0.880 Max: 1.07	Min: 0.839 Avg: 1.11 Max: 1.68	Min: 1.05 Avg: 1.27 Max: 1.73
TigerVNC 1.2 (Compression Level 3)	Min: 1.00 Avg: 1.18 Max: 1.92	Min: 0.555 Avg: 0.756 Max: 0.989	Min: 0.896 Avg: 1.03 Max: 1.48	Min: 1.05 Avg: 1.23 Max: 1.58
TigerVNC 1.2 (Compression Level 6)	Min: 1.00 Avg: 1.19 Max: 1.95	Min: 0.220 Avg: 0.451 Max: 0.973	Min: 0.927 Avg: 0.994 Max: 1.22	Min: 0.970 Avg: 1.14 Max: 1.71
TigerVNC 1.2 (Compression Level 9)	Min: 1.00 Avg: 1.20 Max: 1.96	Min: 0.0274 Avg: 0.0917 Max: 0.910	Min: 0.921 Avg: 1.03 Max: 1.25	Min: 0.866 Avg: 1.02 Max: 1.85

One note about this chart: Since TurboVNC always uses Zlib Compression Level 1 with JPEG encoding, the left two columns allow one to gauge the effectiveness of increasing the level of Zlib compression in TigerVNC 1.2, since the TurboVNC baseline was the same for those tests. The right two columns represent an "apples to apples" regression test, in which each compression level from TigerVNC 1.2 was compared with the equivalent setting in TigerVNC 1.1 (blue cells are apples-to-apples comparisons.)

With Compression Level 1, the palette threshold of 24 from TurboVNC was maintained. It was known from the prior study that increasing this palette threshold to 96 would shift the balance more toward a higher compression ratio and lower throughput, and experimentation with the modified TigerVNC encoder confirmed this. Thus, a palette threshold of 96 was used for compression levels greater than 1.

The first thing to note is that, to within a margin of 2%, the new encoder with Compression Level 1 produced identical output to the TurboVNC encoder. The throughput at Compression Level 1 was centered at 6% faster than the TurboVNC baseline, and the datasets that made heavy use of indexed color encoding ran as much as 16% faster than they did under TurboVNC.

Relative to TigerVNC 1.1, the throughput was better across the board. At the lower compression levels, certain datasets did not compress quite as well as they did under TigerVNC 1.1. In almost all of those cases, however, simply increasing the compression level by one notch restored the performance

and compression ratio that those datasets achieved under TigerVNC 1.1. The exception to this was the CATIA dataset. This dataset has quite a bit of high-frequency content, which compresses poorly with JPEG, but it also has a somewhat high color depth, so the subrectangles it generates don't often fall below the palette threshold of 24. Boosting the compression level to 2 improved the compression ratio for this dataset dramatically (from 52% of the TigerVNC 1.1 baseline to 80%), but in order to restore the compression ratio that this dataset achieved under TigerVNC 1.1 (while still retaining similar levels of performance), it was necessary to turn off JPEG encoding (see Section 3.3 for a discussion of that mode.)

Another thing to note is that, in almost all cases, compression levels higher than 2 are useless with the new TigerVNC 1.2 encoder. There were virtually no gains in compression ratio beyond this level.

3.2 Medium Quality JPEG

The analysis was repeated for Medium Quality JPEG, using a JPEG quality level of 77 with 4:2:2 subsampling in TurboVNC and the equivalent (JPEG Quality Level 5) in the TigerVNC 1.1 and 1.2 encoders. The results appear below.

TigerVNC 1.1 vs. TurboVNC: 4:2:2 Subsampling, JPEG Quality = 77

Dataset	Relative Compression Ratio (vs. TurboVNC Baseline)	Speedup (vs. TurboVNC Baseline)
TigerVNC 1.1 (Compression Level 1)	Min: 0.353 Avg: 0.646 Max: 1.26	Min: 0.320 Avg: 0.612 Max: 0.916
TigerVNC 1.1 (Compression Level 2)	Min: 0.541 Avg: 0.834 Max: 1.37	Min: 0.315 Avg: 0.642 Max: 0.940
TigerVNC 1.1 (Compression Level 3)	Min: 0.725 Avg: 1.00 Max: 1.43	Min: 0.294 Avg: 0.565 Max: 0.879
TigerVNC 1.1 (Compression Level 6)	Min: 0.984 Avg: 1.17 Max: 1.62	Min: 0.225 Avg: 0.358 Max: 0.792
TigerVNC 1.1 (Compression Level 9)	Min: 0.959 Avg: 1.15 Max: 1.63	Min: 0.0288 Avg: 0.0756 Max: 0.575

Here, as with the previous baseline tests, none of the modes could come close to matching TurboVNC on both compression ratio and throughput, and compression levels higher than 6 proved to be useless.

The tests were then re-run with the new TigerVNC 1.2 encoder.

TigerVNC 1.2 vs. TurboVNC and TigerVNC 1.2 vs. TigerVNC 1.1: 4:2:2 Subsampling, JPEG Quality = 77

Dataset	Relative Compression Ratio (vs. TurboVNC Baseline)	Speedup (vs. TurboVNC Baseline)	Relative Compression Ratio (vs. Same Compression Level in TigerVNC 1.1)	Speedup (vs. Same Compression Level in TigerVNC 1.1)
TigerVNC 1.2 (Compression Level 1)	Min: 0.990 Avg: 1.00 Max: 1.01	Min: 0.924 Avg: 1.06 Max: 1.22	Min: 0.795 Avg: 1.55 Max: 2.83	Min: 1.27 Avg: 1.74 Max: 3.19
TigerVNC 1.2 (Compression Level 2)	Min: 1.00 Avg: 1.12 Max: 1.45	Min: 0.601 Avg: 0.871 Max: 1.14	Min: 0.998 Avg: 1.35 Max: 1.86	Min: 1.09 Avg: 1.36 Max: 1.91
TigerVNC 1.2 (Compression Level 3)	Min: 1.00 Avg: 1.15 Max: 1.60	Min: 0.504 Avg: 0.724 Max: 1.01	Min: 0.980 Avg: 1.15 Max: 1.48	Min: 1.05 Avg: 1.28 Max: 1.71
TigerVNC 1.2 (Compression Level 6)	Min: 1.00 Avg: 1.17 Max: 1.78	Min: 0.234 Avg: 0.405 Max: 0.905	Min: 0.943 Avg: 1.00 Max: 1.12	Min: 0.969 Avg: 1.13 Max: 1.72
TigerVNC 1.2 (Compression Level 9)	Min: 1.00 Avg: 1.18 Max: 1.81	Min: 0.0285 Avg: 0.0769 Max: 0.809	Min: 0.934 Avg: 1.03 Max: 1.13	Min: 0.849 Avg: 1.02 Max: 1.88

The results here are similar to those in the previous section. TigerVNC 1.2 with Compression Level 1 now performed almost identically to TurboVNC, and it was faster on the datasets that were encoded using a great deal of indexed color subrectangles.

Relative to TigerVNC 1.1, only with Compression Level 1 did any datasets compress significantly less efficiently. It was our old friend CATIA, and bumping the compression level up to 2 restored the performance that that dataset achieved under TigerVNC 1.1. The throughput relative to TigerVNC 1.1 was better across the board.

3.3 Lossless

The "Tight to Turbo" report primarily discussed the design of the JPEG-based encoding methods in TurboVNC 0.5, but as part of the same study that was conducted in 2008, two additional encoding methods were designed: "Lossless Tight" and "Lossless Tight + Zlib." Lossless Tight is basically the "Lazy Tight" encoding method with no Zlib compression or smoothness detection. Thus, the only form of compression it provides is through indexed color encoding of subrectangles with low numbers of unique colors. The main raison d'etre of Lossless Tight is that it uses less CPU time than Hextile but still provides a compression ratio that is in the ballpark of the latter (around 4-5, on average.) Lossless Tight + Zlib adds Zlib compression and is designed to provide more of a balance between CPU time and compression ratio, so that it at least performs reasonably well on wide-area network connections

(Lossless Tight + Zlib is the encoding method used by TurboVNC's lossless refresh feature.)

The "Lossless Tight" encoding methods use raw subrectangles instead of JPEG, and thus the palette threshold has to be set such that indexed color subrectangles are sent more often than they would be if JPEG was enabled. Experiments in 2008, some of which were repeated during the course of gathering data for this report, showed that maintaining the maximum color divisors from the original TightVNC encoder worked better than using a hard-coded palette threshold.

TigerVNC 1.1, with Compression Level 0 and JPEG disabled, was compared to TurboVNC's "Lossless Tight" encoding method. For all other compression levels, TigerVNC 1.1 was compared to TurboVNC's "Lossless Tight + Zlib" encoding method. As with the previous charts, the blue cells indicate apples-to-apples comparisons, whereas the other tests are meant to measure the usefulness of increasing the level of Zlib compression against a static baseline.

TigerVNC 1.1 vs. TurboVNC: Lossless

Dataset	Relative Compression Ratio (vs. TurboVNC Baseline)	Speedup (vs. TurboVNC Baseline)
TigerVNC 1.1 (Compression Level 0)	Min: 0.611 Avg: 1.30 Max: 1.80	Min: 0.0826 Avg: 0.384 Max: 0.847
TigerVNC 1.1 (Compression Level 1)	Min: 0.381 Avg: 0.985 Max: 1.10	Min: 0.679 Avg: 0.852 Max: 1.04
TigerVNC 1.1 (Compression Level 2)	Min: 0.540 Avg: 1.03 Max: 1.13	Min: 0.748 Avg: 0.834 Max: 1.00
TigerVNC 1.1 (Compression Level 3)	Min: 0.555 Avg: 1.04 Max: 1.22	Min: 0.632 Avg: 0.760 Max: 0.867
TigerVNC 1.1 (Compression Level 6)	Min: 0.975 Avg: 1.06 Max: 1.60	Min: 0.292 Avg: 0.433 Max: 0.696
TigerVNC 1.1 (Compression Level 9)	Min: 0.985 Avg: 1.08 Max: 1.61	Min: 0.0457 Avg: 0.116 Max: 0.539

In this case, it was observed that compression levels greater than 2 were useless. The performance of Compression Levels 0 and 1 was mixed. Some datasets compressed more efficiently with TigerVNC 1.1, but in most cases, its throughput relative to TurboVNC was significantly less.

Note that TurboVNC bypasses Zlib completely when using the Lossless Tight encoding method, so that is one factor in the large disparity between its throughput and the throughput of TigerVNC 1.1 at Compression Level 0. The reason that TurboVNC bypasses Zlib is that, even when using Zlib

Compression Level 0 (no compression), Zlib adds a significant amount of performance overhead.

TigerVNC 1.2 vs. TurboVNC and TigerVNC 1.2 vs. TigerVNC 1.1: Lossless

Dataset	Relative Compression Ratio (vs. TurboVNC Baseline)	Speedup (vs. TurboVNC Baseline)	Relative Compression Ratio (vs. Same Compression Level in TigerVNC 1.1)	Speedup (vs. Same Compression Level in TigerVNC 1.1)
TigerVNC 1.2 (Compression Level 0)	Min: 0.988 Avg: 0.999 Max: 1.00	Min: 0.0664 Avg: 0.334 Max: 1.00	Min: 0.554 Avg: 0.766 Max: 1.63	Min: 0.796 Avg: 0.869 Max: 1.26
TigerVNC 1.2 (Compression Level 1)	Min: 0.991 Avg: 1.00 Max: 1.04	Min: 0.800 Avg: 0.996 Max: 1.36	Min: 0.912 Avg: 1.02 Max: 2.65	Min: 1.01 Avg: 1.17 Max: 2.00
TigerVNC 1.2 (Compression Level 2)	Min: 1.01 Avg: 1.03 Max: 1.23	Min: 0.788 Avg: 0.909 Max: 1.27	Min: 0.924 Avg: 1.00 Max: 1.95	Min: 0.926 Avg: 1.09 Max: 1.56
TigerVNC 1.2 (Compression Level 3)	Min: 1.01 Avg: 1.04 Max: 1.58	Min: 0.713 Avg: 0.842 Max: 1.05	Min: 0.930 Avg: 1.00 Max: 1.93	Min: 0.904 Avg: 1.11 Max: 1.57
TigerVNC 1.2 (Compression Level 6)	Min: 0.985 Avg: 1.08 Max: 1.76	Min: 0.311 Avg: 0.497 Max: 0.733	Min: 0.995 Avg: 1.02 Max: 1.10	Min: 1.01 Avg: 1.15 Max: 1.67
TigerVNC 1.2 (Compression Level 9)	Min: 0.986 Avg: 1.10 Max: 1.80	Min: 0.0459 Avg: 0.120 Max: 0.545	Min: 1.00 Avg: 1.02 Max: 1.12	Min: 0.915 Avg: 1.04 Max: 1.45

Compression Levels 0 and 1 now produced almost identical output to TurboVNC's Lossless Tight and Lossless Tight + Zlib encoding methods (respectively.) Compression Level 0 fell considerably short of the TurboVNC performance target because of the overhead of passing the uncompressed images through Zlib. A possible future project would be to figure out how to enhance Zlib such that it could pass through uncompressed data without overhead when using Zlib Compression Level 0. It would also be straightforward to add TurboVNC's Zlib bypass feature to TigerVNC, but since that feature uses an unauthorized extension to the RFB protocol, the former approach is cleaner. Then again, as CPUs become faster, the relevance of the Lossless Tight encoding method may not be enough to be worthy of additional effort.

Returning to the chart above, it should be noted that only very few datasets benefited from compression levels higher than 1. Those datasets that benefited were the ones that were encoded using more indexed color rectangles. Even these datasets experienced compression ratio gains of only 20-25% when moving from Compression Level 1 to Compression Level 6, at the expense of losing 2/3 of their throughput.

The only significant performance regressions of note were under Compression Level 0. Because this mode really needs some form of Zlib bypass, its usefulness will be somewhat limited until/unless that feature is implemented.

Compression Level 1 under the new encoder was shown to be an adequate replacement for "Lossless Tight + Zlib", and it can thus be used as the basis for the future implementations of a lossless refresh feature in TigerVNC.

3.4 Conclusions and Recommendations

- In the new encoder, compression levels higher than 2 are of limited usefulness, and in no case, either in the old or new encoder, could a compression level higher than 6 be shown to have any benefit whatsoever.
    - Recommendation 1: Set the default compression level to 1.
    - Recommendation 2: Disallow levels higher than 6 from being selected in the GUI (they could still be selected from the command line.)
    - Recommendation 2: Implement a compression level slider or a dial in the TigerVNC Viewer GUI which warns the user of the limited effectiveness of levels 4, 5, and 6, perhaps using color coding (a green zone and a red zone, for instance.)
- The new encoder with JPEG enabled and Compression Level 1 produces similar output to TurboVNC's JPEG-based encoding methods.
- The new encoder with JPEG disabled and Compression Level 1 produces similar output to TurboVNC's Lossless Tight + Zlib encoding method.
- The new encoder with JPEG disabled and Compression Level 0 produces similar output to TurboVNC's Lossless Tight encoding method, but the new mode in TigerVNC needs either a Zlib bypass mechanism or optimizations to the Zlib library to achieve the same performance as Lossless Tight. At the moment, the mode is of limited usefulness without those optimizations. Feedback from the user community would be useful in ascertaining whether optimizing this mode is desirable or whether Hextile and Raw are adequate.