Supplemental Material:

ManiVault: A Flexible and Extensible Visual Analytics Framework for High-Dimensional Data

Alexander Vieth[*1], Thomas Kroes[*2], Julian Thijssen[*2], Baldur van Lew[2], Jeroen Eggermont[2], Soumyadeep Basu[2], Elmar Eisemann[1], Anna Vilanova[3], Thomas Höllt[•1], Boudewijn Lelieveldt[•1,2]

S1: Benchmarks

Speed ManiVault can show progressive updates of analytics plugins with only small additional computational penalties. To show this, we compute t-SNE embeddings with a ManiVault analytics plugin that uses the HDI library GPGPU implementation of t-SNE [4, 5]. First, we compute embeddings non-progressively, and then, in a second setting, we show intermediate embeddings every 10 gradient descent iterations (respectively "no updated" and "with updates" in Tab 2. Additionally, we compare these runs with a lightweight python wrapper [7] around the same t-SNE library. Every embedding is laid out over 500 gradient descent iterations. The non-progressive computation is slightly faster than the Python wrapper around the same library calls. The difference between the total runtime of the t-SNE embeddings in ManiVault with and without updates is explained by the difference in the gradient descent time: In the former setting, the analytics plugin notifies ManiVault's core about the current embedding layout. All measurements were taken on a machine equipped with an NVIDIA GeForce RTX 2080 SUPER GPU and an Intel Core i5-9600K CPU and running Windows 11 22H2. The supplemental material "benchmark_time.xlxs" contains the full data.

Supplementary Table 2: Duration of t-SNE embedding computations with the same implementation, invoked via a Python wrapper and ManiVault, once showing only the final embedding and once progressively updating a scatterplot. Times, in seconds, are averages over 10 runs with sample standard deviation.

Data set	Swiss Roll 3D [6]	COIL-20 [3]	MNIST [2]	Fashion-MNIST [8]	10x Mouse [9]
# points	1,500	1,440	70,000	70,000	1,306,127
# dimensions	3	16,384	784	784	50 (first PCs)
nptsne [7] (Python wrapper)	0.30 (0.02)	2.32 (0.08)	23.31 (0.14)	20.58 (0.01)	268.38 (2.21)
ManiVault (no updates)	0.58 (0.05)	2.37 (0.05)	22.51 (0.11)	20.20 (0.27)	258.60 (5.76)
ManiVault (with updates)	0.59 (0.07)	2.46 (0.09)	22.85 (0.15)	20.24 (0.11)	257.91 (4.02)

Note: Times in seconds, sample standard deviation in parentheses.

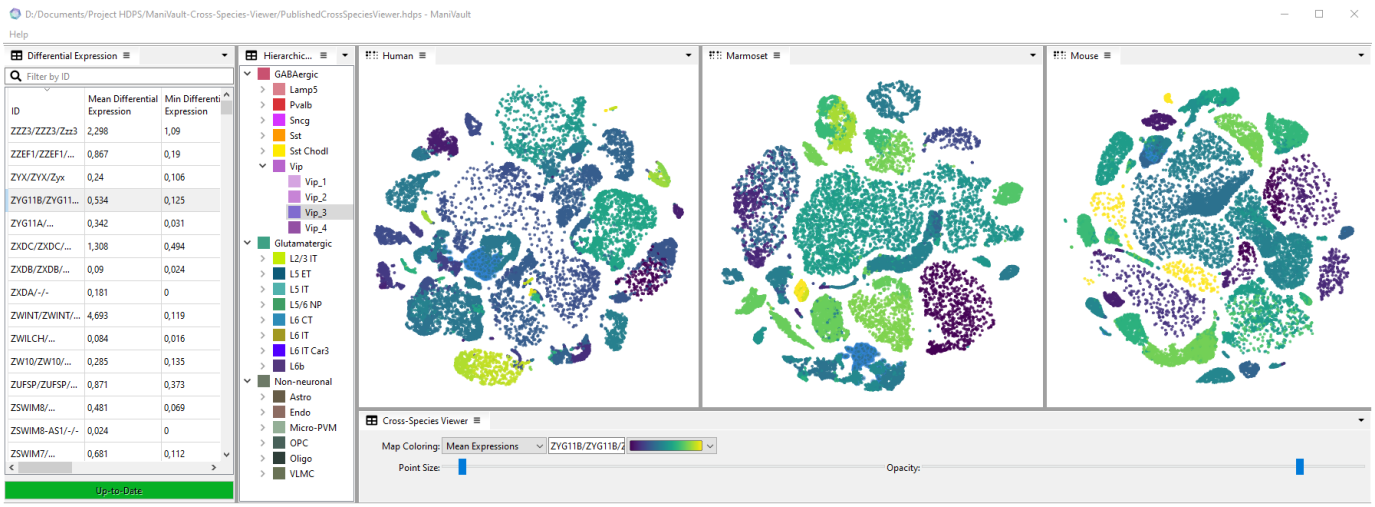
Memory After starting ManiVault, with the Data Hierarchy and Data Property Viewer open, the software consumes around 87 MB of memory (on Windows). Loading data sets comes with a small memory overhead. Here, we loaded various data sets, as listed in Tab 2, and compared their binary size on disk with the growing memory footprint of ManiVault after loading them. We observe a $0.7 - 1.5$ MB overhead per data set, compared to their binary size, when utilizing the point data type plugin. For larger data set, it can be useful to trade off precision for lower memory uptake. We can employ a bfloat16 floating point implementation [1] to store large data set, and thereby effectively half the memory ManiVault requires: e.g. the 10x Mouse data will take up 126.15 MB instead of 249.87 MB. The supplemental material "benchmark_memory.xlxs" contains the full data.

Supplementary Table 3: Memory consumption of loaded data sets, as listed in Tab 2, in ManiVault compared to their binary size on disk. Values are averages over 4 loaded data sets.

Data set	[6]	[3]	[2]	[8]	[9]
Binary type	float32	uint8	uint8	uint8	float32
Raw binary	0.017	22.5	52.34	52.34	249.12
ManiVault (float32)	0.97	-	-	-	249.87
ManiVault (uint8)	-	23.77	53.5	54.1	-

Note: Values in MB. Slight deviations might occur due to Qt's memory management on Windows 11, e.g., the difference between MNIST and Fashion-MNIST.

S2: Larger Figures



Supplementary Fig. 12: Large version of Fig. 11.

Supplemental References

[1] N. Dekker. biovault_bfloat16. github.com/biovault/biovault_bfloat16, 2020.

[2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. Access: yann.lecun.com/exdb/mnist. doi: 10.1109/5.726791

[3] S. A. Nene, S. K. Nayar, and H. Murase. Columbia Object Image Library (COIL-20). Technical Report CUCS-005-96, Department of Computer Science, Columbia University, 1996. Access: cs.columbia.edu.

[4] N. Pezzotti. High dimensional inspector. github.com/Nicola17/High-Dimensional-Inspector, 2018. doi: 10.5281/zenodo.1303855

[5] N. Pezzotti, J. Thijssen, A. Mordvintsev, T. Höllt, B. V. Lew, B. P. Lelieveldt, E. Eisemann, and A. Vilanova. GPGPU linear complexity t-SNE optimization. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):1172–1181, 2020. doi: 10.1109/tvcg.2019.2934307

[6] J. Tenenbaum. Mapping a Manifold of Perceptual Observations. In M. Jordan, M. Kearns, and S. Solla, eds., *Proc. NIPS*, vol. 10, p. 682–688. MIT Press, 1997. Access: scikit-learn.org.

[7] B. van Lew. nptsne. github.com/biovault/nptsne, 2021. doi: 10.5281/zenodo.5801124

[8] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. Access: github.com/zalandoresearch/fashion-mnist, 2017. arXiv preprint. doi: 10.48550/arXiv.1708.07747

[9] G. X. Y. Zheng, J. M. Terry, and J. H. Bielas. Massively Parallel Digital Transcriptional Profiling of Single Cells. *Nature Communications*, 8(1):14049, 2017. Access: file.biolab.si/opentsne/benchmark. doi: 10.1038/ncomms14049