Editors: Cláudio T. Silva, csilva@cs.utah.edu
Joel E. Tohline, tohline@lsu.edu

USING PYTHON FOR SIGNAL PROCESSING AND VISUALIZATION

By Erik W. Anderson, Gilbert A. Preston, and Claudio T. Silva

Applying Python to a neuroscience project let developers put complex data processing and advanced visualization techniques together in a coherent framework.

An ever-increasing number of scientific studies are generating larger, more complex, and multimodal datasets. As a result, data analysis tasks are becoming more demanding. To help tackle these new challenges, more disciplines must now incorporate advanced visualization techniques into their standard data processing and analysis methods. While many systems let scientists explore, analyze, and visualize their data, such solutions are often domain specific, limiting their scope as general processing tools. One way to enhance their flexibility is to build them on top of an interpreted language.

The Python programming language[1] provides a development environment suitable to both computational and visualization tasks. One of Python's key advantages is that it lets developers use packages that extend the language to provide advanced capabilities, such as array and matrix manipulation,[2] image processing,[3] digital signal processing,[2] and visualization.[4] Several popular data exploration and visualization tools have been built in Python, including VisIt (www.llnl.gov/visit), Paraview (www.paraview.org), climate data analysis tools (CDAT; www2-pcmdi.llnl.gov/cdat), and VisTrails (www.vistrails.org).

In our work, we use VisTrails; however, nearly any Python-enabled application can produce similar results. The neuroscience field often uses both multimodal data and computationally complex algorithms to analyze data collected from study participants. Here, we investigate a study in which magnetic resonance imaging (MRI) is combined with electroencephalography (EEG) data to examine working memory.

A Neuroscience Example

MRIs provide a 3D depiction of the brain's structure, which is a natural spatial organization for the EEG sensors. EEG data is collected from 64 sensors placed on the scalp. These sensors measure voltages at the scalp generated by brain activity. Generally, researchers view working memory as the short-term information retention and its integration into the executive decision-making process. Research has shown that brain activity regarding working memory is spectrally and spatially organized.[5] Specifically, we can assess working memory performance by measuring changes to the energy densities, phase relationships, and frequency shifts in the alpha band of frequencies (7–13 hertz), located in the dorsal-lateral prefrontal cortex.

To analyze working memory's temporal and spectral organization, researchers use advanced signal processing techniques to transform the time series gathered from EEG sensors in specific locations into Fourier-based representations. As the sensor locations are known, spatial representations of various EEG-based quantities are implied, but difficult to see.

Combining MRI data with the known EEG sensor locations highlights spatial relationships between the sensors and the activity in the brain they measure. During more complex analysis, researchers often use MRI data to determine participant-specific finite element meshes for use in solving the inverse problem; this assigns scalar values to the cortical surface using data collected at sensor locations. This method of data interpolation is more robust than those from interpolation schemes. Solving the inverse problem in this way is useful for tasks such as source localization for the identification of potential epileptogenic regions in the brain. Fortunately, for determining spatial relationships between active brain regions and EEG sensors, radial basis function interpolation has proven to be a good approximation.[6] Regardless of the method we use to map scalars from discrete point locations to a surface representation, we must fuse data collected from different acquisition methods into a coherent representation. However, to achieve this type of data fusion, we first must register the MRI data with the EEG sensor locations. Furthermore, we must segment the MRI to extract the brain surface so we can use it to map the scalar values derived from EEG.

Our study culminates in a visualization depicting the spectral power at the alpha band of frequencies (7–13 Hz) across the entire cortical surface. Additionally, the time-frequency representation for each sensor's data must be selectable to provide details when queried. This task requires the use of signal processing techniques to decompose the EEG data, image processing techniques to segment the brain surface from the MRI, interpolation techniques to approximate the brain surface activity responsible for the EEG data, as well as 1-, 2-, and 3D rendering techniques for data presentation.

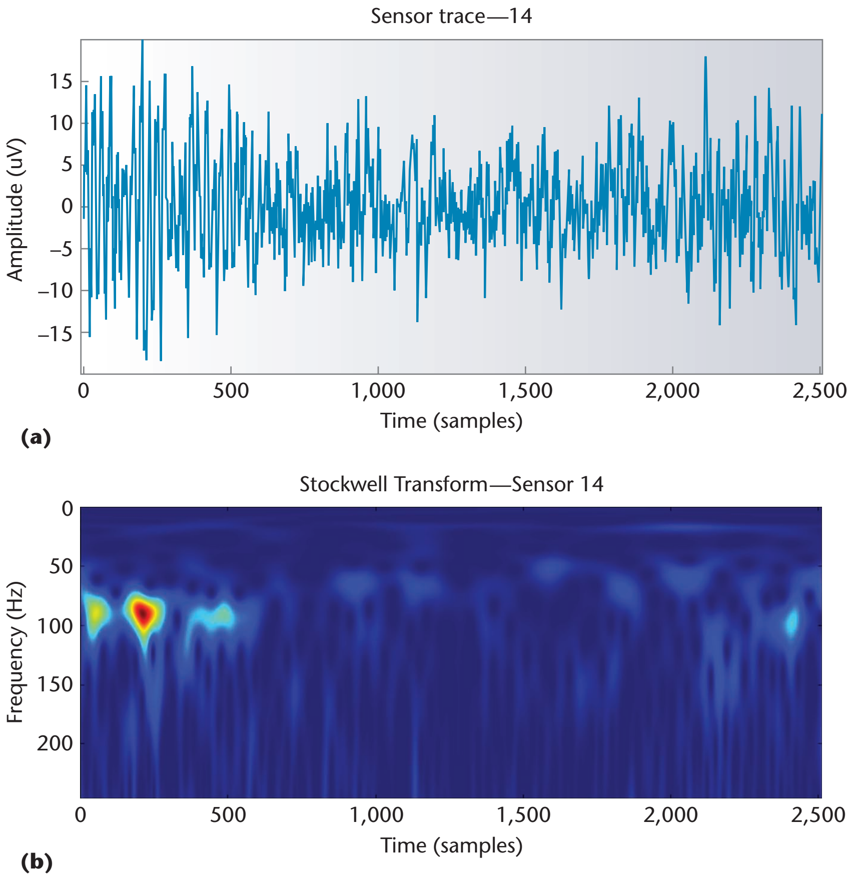

Figure 1. An example Stockwell transform. (a) A plot of a single sensor's raw data trace. (b) The S-transformed representation of the sensor's raw data. Because this visualization shows all representable frequencies at each timestep, it allows analysis of the time series' frequency evolution.

EEG Signal Processing

To process EEG data for interpretation and further analysis, we can use Fourier-based transforms to determine spectral properties of brain activity. Determining how spectral properties change over time is important to the study of working memory. We need these techniques because we assess working memory performance using changes in specific spectral properties of the EEG signals measured at the scalp. When working memory is tasked, alpha-band power increases while simultaneously shifting to slightly higher frequencies. The SciPy Python package includes fast Fourier transforms (FFTs), various filters, and some wavelet implementations that are useful in the analysis of working memory. FFT computation is fast within SciPy because it uses the FFT implementation in the FFTW libraries.[7] However, the global nature of FFTs make it inadequate for analyzing spectral content's evolution over time.

When examining the temporal aspects of a time series' frequency-based representations, several different transformations are applicable. One-time frequency decomposition, short-time Fourier transforms (STFTs) use time windows to examine spectral evolution. However, STFTs suffer from a uniform packing of the time-frequency space. Alternatively, wavelet-based approaches use an adaptive resolution scheme to pack the time-frequency space, but lack a direct mapping from scale to frequency. On the other hand, the Stockwell transform[8] uses an adaptive resolution scheme similar to wavelet transforms, but still maintains a direct mapping to the frequency domain. Figure 1 shows the results of a Stockwell transform representing the energy density between 1 and 250 Hz during an experiment.

We chose to use Stockwell transforms throughout our analysis. Because other time-frequency representations can be used to analyze this dataset, the processing pipeline must be easily changeable to accommodate different Fourier-based decompositions. Such flexibility is important when exploring not just time series data, but the different data products resulting from processing and analysis. Using Python as the underlying data analysis framework lets users easily change analyses on the fly using various implementations, from user-created specifications to robust, compiled libraries.

Unfortunately, Stockwell transforms are computationally intensive and, in this situation, compiled languages generally perform better than interpreted languages. To accelerate the transform's computation, we implemented it in optimized C (available at http://kurage.nimh.nih.gov/meglab/Meg/Stockwell).

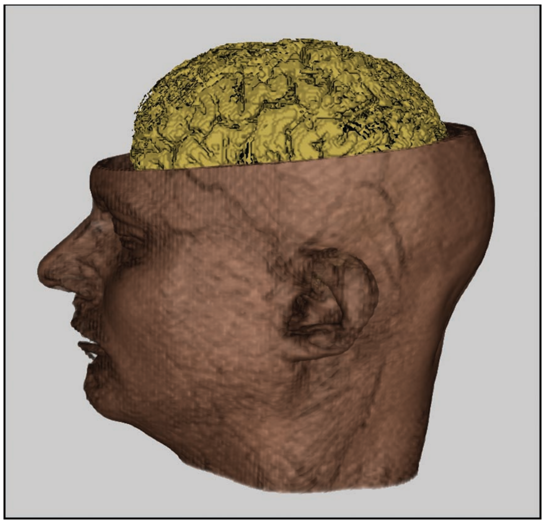
Figure 2. Automatic segmentation results. Registering sensor locations with MRI-based data reveals the spatial relationships between the EEG sensors and the structures evident in the MRI volume.

This small C library is made accessible to Python through compiled bindings, which let Python use methods written in C or C++. These bindings can also be generated in Python directly using the Ctypes package (http://docs.python.org/library/ctypes.html). Wrapping methods with Python bindings lets users perform code execution in faster compiled languages, lending additional speed to Python-based applications. This execution method makes computationally intense algorithms tractable inside an interpreted language environment.

Volume Segmentation and Registration

Once EEG data is processed and analyzed, we must extract structural information from the MRI volume collected. Segmenting such volumes is a difficult task, best suited for specialty libraries and algorithms.[9] We've employed the Insight toolkit (ITK)[3] to extract the brain area from the MRI data. ITK is a collection of image processing algorithms implemented in C++ that provide volumetric operations from smoothing to segmentation. As with many other libraries, ITK is distributed with a collection of Python bindings. In ITK's case, we can do this automatically using Kitware's CableSwig, which provides a mechanism to wrap highly templated C++ libraries for use with Python. Without CableSwig, ITK's templated nature would generate an unnecessarily large number of bindings, complicating its use in Python.

We're using ITK here because we need to segment the MRI volume. To simplify this procedure, we decided to use Marcel Prastawa's automatic cortical extraction method.[10] After segmentation, we extracted an iso-surface of the cortex to augment the MRI visualization by embedding the surface in a raditional volume rendering. Figure 2 shows the result of this automatic segmentation. Combining the clipped MRI volume with the extracted cortical surface reveals spatial relationships between different structures.

Visualizations of this nature make it easier to analyze the data. In particular, such multimodal visualizations highlight the correlations between the brain structures' spatial relationships and the EEG's brain activity measurements. One way to visualize spatial relationships is to use topographic maps (topomaps). This method uses a surrogate representation of the scalp and places values on it based on the EEG data collected. Figure 3 shows two timesteps in the experiment, but doesn't account for the MRI data's 3D nature. By registering the MRI volume, cortical surface, and sensor locations provided by the EEG manufacturer, we can form a cohesive representation of the entire dataset respecting the individual structures measured by the MRI. Figure 4 shows the dataset after performing the 3D registration of the MRI volume and sensor locations.

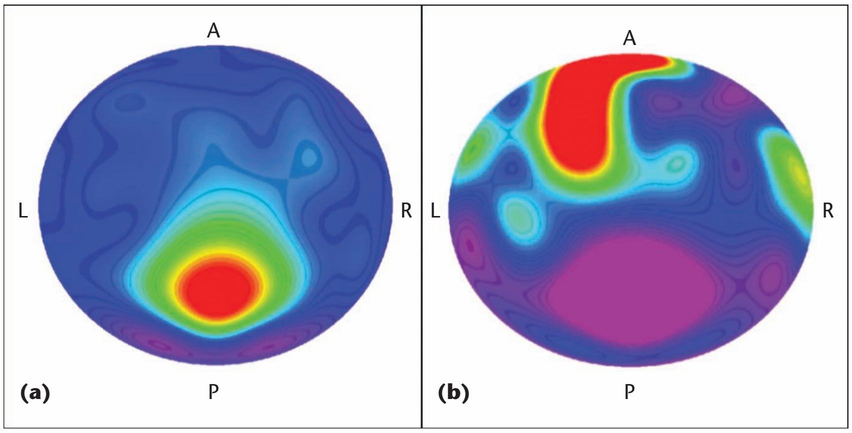

Figure 3. Topographic maps generated from EEG data projected onto a surrogate head provide some spatial indications of brain activity. Here, the topomaps are colored based on average alpha power (a) when a participant is in a resting state and (b) during a working memory task. The surrogate head model represents only data collected and interpolated onto the scalp, making no attempt to map values to the cortical surface.

Visualization

For our visualization tasks, we use the VTK library, a C++ library that

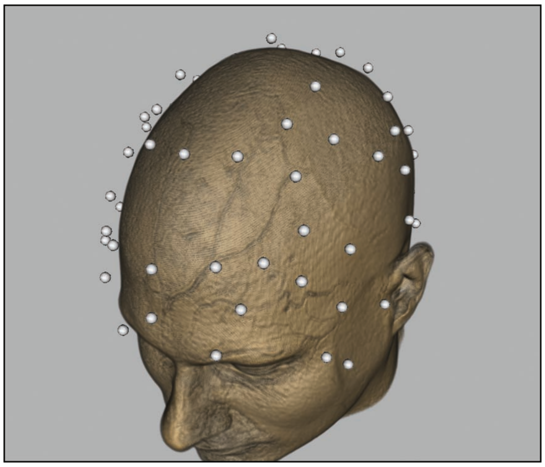

Figure 4. The dataset after 3D registration of the MRI volume and sensor locations. Registering sensor locations with MRI-based data reveals the spatial relationships between the EEG sensors and the structures present in the MRI volume.

provides advanced visualization capabilities to Python programs. This lets us combine volume rendering, surface rendering, and point rendering for the MRI volume, cortical surface, and registered sensor locations, respectively. However, we must perform additional processing to properly map scalar values onto the cortex.

To do this, we use radial-basis function (RBF) interpolation, which is available in SciPy and lets us approximate scalar values on the cortical surface given values at each sensor location. Although this method isn't equivalent to a solution of the inverse problem, it provides a good estimate of brain activity's spatial organization. Figure 5 shows the results of applying RBF interpolation for the average power in the alpha band of frequencies (7–13 Hz) measured at each sensor location.

VTK also provides functionality for selection and on-the-fly updates to enhance interaction. This mechanism lets us select individual sensors and display their unique time-frequency representations. Instead of displaying the 2D time-frequency planes using VTK, we use the Pylab plotting functionality distributed with SciPy. Pylab specializes in 1- and 2D plotting techniques and proves to be an ideal rendering library for time-frequency data. Figure 4a shows 3D data rendered using VTK, while Figure 4b shows results from using Pylab to visualize the selected sensors' time-frequency planes.

Discussion

Tools allowing rapid exploration of large and multimodal datasets are more important than ever in scientific research. Interpreted languages such as Python provide a solid foundation for developing powerful yet flexible data analysis and visualization tools. However, flexibility of analysis and visualization must be combined to enhance the exploration process. We implemented our Python system as a series of VisTrails Python modules.[12] The VisTrails system is a visual programming paradigm that represents computational elements as drag-and-drop modules that users connect together to form programs. The drag-and-drop system makes it easy to replace functionally equivalent computations, such as replacing an STFT with a Stockwell transform.

Providing a tool that supports flexible visualization and analysis lets scientists draw more insightful conclusions. Additionally, the ability to change analysis techniques lets them gain important insights more quickly. Using visual programming paradigms such as VisTrails also makes changing analysis techniques easier for nonprogrammers, facilitating the tool's use for insight generation.

In addition to making changes to analysis techniques easier, Python has also proven to be excellent at combining related yet disparate data into a single, useful representation. Figure 5 exemplifies this data fusion, successfully combined structural volumetric data with processed time series data. As Figure 6 shows, when animated, visualizations show not just the brain's activity, but also how it evolves

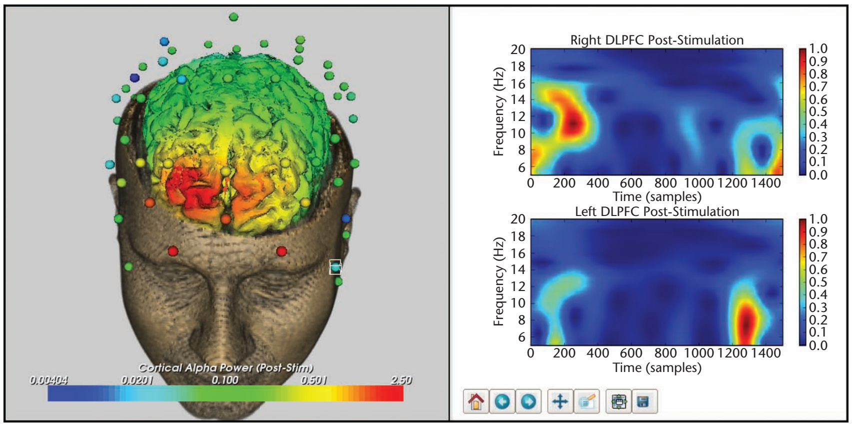
Cortical Alpha Power (Post-Stim)
0.00404
0.0201
0.100
0.501
2.50
Right DLPFC Post-Stimulation
Frequency (Hz)
Time (samples)
Left DLPFC Post-Stimulation
Frequency (Hz)
Time (samples)

Figure 5. Interactive visualizations of multimodal data based on data processing and visualization in Python. (a) Alpha-band activity (7–13 Hz) in the brain's frontal regions. (b) Specific time-frequency representations determined by the user-driven selection of EEG sensors. The result is a pattern consistent with those described in literature.[12]

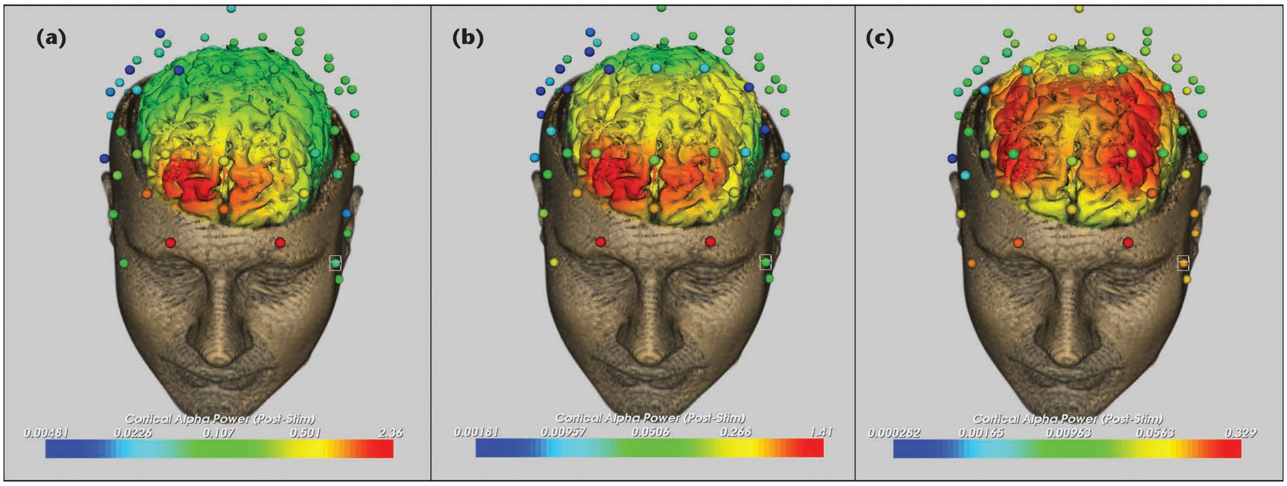

Figure 6. Rendering multiple time-steps of this data depicts the underlying neural circuit associated with working memory. In this case, activity spreads throughout the cortex moving between (a) the dorso-lateral pre-frontal cortex, (b) the temporal region, and (c) the parietal areas. This series of animated images highlights not just the spectral and temporal organization of working memory, but also it's spatial coherence. Without adequate visualization techniques, the link between memory and specific brain regions would be more difficult to determine.

spectrally and spatially throughout the cortex. Spatial data analysis is common in neuroscience, but most techniques respect spatial relationships by letting scientists select groups of sensors located at specific places to focus their analysis efforts. Providing methods that let neuroscientists examine their dataset as a whole lets them perform insightful analysis more quickly. The efficiency gained by better utilizing visualization as a tool lets them identify new and unexpected behavior more easily.

We've presented a small case study of Python's use as a foundation for exploring and analyzing multimodal data. Python's large user community and array of libraries enhance the language by providing new functionality useful in every aspect of data processing and management. The language's availability, flexibility, and ease of use facilitate scientific endeavors from computationally intense applications to collaborative analysis.

The Python programming language provides a strong foundation for building flexible applications. Leveraging optimized C, C++, and even FORTRAN libraries by wrapping them for use in Python permits flexible and powerful applications. Additional open source libraries supported by Python, such as Qt (www.qt.nokia.com), provide powerful user interface capabilities for any application.

Python isn't alone in its use of compiled libraries. Other interpreted languages also take advantage of external libraries to overcome execution speed barriers. Probably the most notable of these languages is Matlab. Although Matlab and Python are largely equivalent in terms of their capabilities, Python is open source, making it an attractive solution to many applications. Applications developed in Python can be widely distributable, making it easier to enable collaboration among scientists at various locations.

Working versions and source code for the material we've presented here are available at www.vistrails.org. CiSE

Acknowledgments

The US National Science Foundation, the US National Institutes of Health, the US Department of Energy, and IBM Faculty Awards supported this work.

References

1. G. van Rossum, *Python Tutorial*, tech. report CS-R956, Center for Mathematics and Computer Science (CWI), Amsterdam, 1995.
2. T.E. Oliphant, "Python for Scientific Computing," *Computing in Science & Eng.*, vol. 9, no. 3, 2007, pp. 10–20.
3. T. S. Yoo et al., "Engineering and Algorithm Design for and Image Processing API: A Technical Report on ITK—The Insight Toolkit," *Proc. Medicine Meets Virtual Reality*, vol. 85, 2002, pp. 586–592.
4. W. Schroeder, K. Martin, and B. Lorensen, *The Visualization Toolkit*, 3rd ed., Kitware Inc., 2004.
5. A. von Stein and J. Sarnthein, "Different Frequencies for Different Scales of Cortical Integration: From Local Gamma To Long Range Alpha/Theta Synchronization," *Int'l J. Psychophysiology*, vol. 38, no. 3, 2000, pp. 301–313.
6. M.R. Nuwer, "Quantitative EEG: Techniques and Problems of Frequency Analysis and Topographic Mapping," *J. Clinical Neurophysiology*, vol. 5, no. 1, 1988, pp. 1–43.
7. M. Frigo and S.G. Johnson, "FFTW: An Adaptive Software Architecture for the FFT," *Proc. IEEE Int'l Conf. Acoustics Speech and Signal Processing* (ICASSP) vol. 3, IEEE Press, 1998, pp. 1381–1384.

8. R.G. Stockwell, "A Basis For Efficient Representation of the S-Transform," *Digital Signal Processing*, vol. 17, no. 1, 2007, pp. 371–393.
9. H. Xue et al., "Automatic Segmentation and Reconstruction of the Cortex from Neonatal MRI," *NeuroImage*, vol. 38, no. 3, 2007, pp. 461–477.
10. M. Prastawa et al., "Automatic Segmentation of MR Images of the Developing Newborn Brain," *Medical Image Analysis*, vol. 9, no. 5, 2005, pp. 457–466.
11. W. Klimesch, "EEG Alpha and Theta Oscillations Reflect Cognitive and Memory Performance: A Review and Analysis," *Brain Research Reviews*, vol. 29, nos. 2–3, 1999, pp. 169–195.
12. L. Bavoil et al., "VisTrails: Enabling Interactive Multiple-View Visualizations," Proc. *IEEE Visualization*, IEEE CS Press, 2005, doi.ieeecomputersociety.org/10.1109/VIS.2005.113.

Erik W. Anderson is a research assistant and PhD candidate at the University of Utah. His research interests include scientific visualization, signal processing, computer graphics, and multimodal visualization. Anderson has a BS in computer science and a BS in electrical and computer engineering from Northeastern University. Contact him at eranders@sci.utah.edu.

Gilbert A. Preston is a clinical and research psychiatrist at Utah State hospital. His research interests include noninvasive brain stimulation, a novel approach for treating cognitive disorders, and the identification of neural frequencies subtending cognitive performance as applied to psychiatric disorders. Preston received his MD from the University of Buffalo. Contact him at gpreston@utah.gov.

Claudio T. Silva is an associate professor at the University of Utah. His research interests include visualization, geometry processing, graphics, and high-performance computing. Silva has a PhD in computer science from SUNY at Stony Brook. He is a member of IEEE, the ACM, Eurographics, and Sociedade Brasileira de Matematica. Contact him at csilva@cs.utah.edu.