Introduction

The purpose of developing yet another web-based image viewer, NiftyView, is to use WebGL to take advantage of the parallel computing power in Graphics Processing Units (GPU) hardware for the acceleration of rendering and processing of medical images in web applications. Although several web-based medical image viewers such as Papaya [https://github.com/rii-mango/Papaya], BrainBrowser [https://brainbrowser.cbrain.mcgill.ca/] and SliceDrop [http://slicedrop.com] are currently available, the slow performance of the web-based applications is still one of the major limitations of web-based image viewers. NiftyView is a free web application developed in JavaScript. It has zero footprint; only a web browser and an Internet connection are needed to run NiftyView. It’s advantageous over conventional desktop applications in that NiftyView doesn’t require installation and constant updates. The current version supports NIfTI [http://nifti.nimh.nih.gov] and DICOM [http://dicom.nema.org] format. As a minimal image viewer, it’s a convenient tool for users who need a quick and easy tool for viewing medical images. Currently, the beta version of NiftyView is freely available [http://www2.hawaii.edu/~weiran/NiftyView.html].

Approach

NiftyView is developed in JavaScript with jQuery [http://jquery.com] for HTML document manipulation and event handling, jQueryUI [http://jqueryui.com] for user interface, and DicomParser [https://github.com/chafey/dicomParser] for parsing DICOM files. It’s compatible with popular web browsers including Internet Explorer, Safari, Firefox, and Opera. Either DICOM or NIfTI files can be loaded by dragging files into the browser window. Loaded images can be displayed in single-slice mode or tiled mode. After loading, images are automatically arranged according to the scan IDs for DICOM files and the file name for NIfTI files, respectively. Current functions include image zooming and adjustment of image brightness and contrast. The number of image columns can be adjusted in the tiled mode to maximize the use of the display space. The contrast and brightness of images can be adjusted by clicking and holding the right mouse button or using a double slider widget in the horizontal tool bar at the top of the window. For proof-of-concept, functions such as pixel windowing and scaling are programmed using WebGL by translating the arithmetic operations in image processing to 3D graphics primitives using WebGL’s programmable shaders. The pixel values of an image are loaded into a frame buffer. A vertex shader is programmed to define vertices corresponding to the coordinates of the image, and a fragment shader is programmed to perform arithmetic operations, which are performed in parallel to a massive number of image pixels.

Results

See Figs. 6 and 7.

Discussion

One of the major limitations of current web-based image viewers is the slow performance compared to their desktop counterparts. There are collective efforts in industry to develop new technologies such as WebAssembly and WebGL to narrow this performance gap. The highly parallel nature in processing image pixels independently allows the use of WebGL to achieve a significant speedup, as shown in this abstract. Currently, there are several similar existing web applications such as Papaya, BrainViewer, and slicedrop.com, which are more mature and offer varieties of features. However, the main goal of the continuing effort in the development of NiftyView is to achieve a high performance for image processing using GPU via WebGL. NiftyView has a minimal boilerplate and can handle a large number of files with relative ease. Future work will be focused on developing a WebGL-enabled version, adding more image processing features, adding support of accessing files stored in HIPAA (Health Insurance Portability and Accountability Act) compliant cloud storage services such as Box and Amazon S3. The stable version of NiftyView will be released under a General Public License that allows end users to freely run, modify, and share the program.

Conclusion

NiftyView is a free and convenient web application for quick and easy viewing of NIfTI and DICOM medical images. We have shown that a factor of six to eight acceleration can be achieved using WebGL for image processing.

Availability of supporting data

More information about this project can be found at: http://www2.hawaii.edu/~weiran/NiftyView.html

Competing interests

None.
A8

Human Connectome Project Minimal Preprocessing Pipelines to Nipype

Eric Earl1, Damion V. Demeter1, Kate Mills1, Glad Mihaï2, Luka Ruzic1, Nick Ketca1, Andrew Reineneberg3, Marianne C. Reddanda1, Anne-Lise Goddings3, Javier Gonzalez-Castillo6, Krzysztof J. Gorgolewski7

1Oregon Health & Science University, Portland, OR, USA; 2University of Greifswald, Greifswald, Germany; 3Duke Institute for Brain Sciences, Durham, NC, USA; 4Department of Psychology and Neuroscience, University of Colorado, Boulder, CO, USA; 5Institute of Cognitive Neuroscience, University College London, London, United Kingdom; 6Section on Functional Imaging Methods, Laboratory of Brain and Cognition, National Institute of Mental Health, Bethesda, MD, USA; 7Department of Psychology, Stanford University, Stanford, CA, USA

Correspondence: Eric Earl (earl@ohsu.edu) – Oregon Health & Science University, Portland, OR, USA

GigaScience 2016, 5(Suppl 1):48

Introduction

The goal was to convert the Human Connectome Project (HCP) Minimal Preprocessing Pipelines into Nipype code.

The HCP minimal preprocessing pipelines [1] represent a significant advance in image processing pipelines in our time. They provide preprocessed volume and surface data in native and atlas space, for both functional and structural data. Nipype is an open source neuroimaging project for designing imaging pipelines which has been around since 2011 and provides many excellent features for provenance and reliability of processing pipelines [2]. Together, these two pieces of software would allow for a more robust, more flexible synergy of pipeline design and operability.

Approach

The first goal was to train the would-be Nipype developers on the Nipype python standards for writing and running interfaces. Once trained, the plan was to implement the HCP scripts into Nipype interfaces from the top-level inward to the sub-level scripts. The secondary goal was to make these sub-level scripts more flexible and require less specific scans to run the pipelines. The collection of nine ultimate pipelines to implement were with or without T1s or T2s and with or without Fieldmap or Reverse-Phase-Encode EPIs as seen in Table 3.

Results

Conceptually these goals sounded reasonable enough to do all HCP scripts at once during the hackathon, but the learning and additional setup time was not accounted for, so the scope of the project was too big for two days of on and off coding, even among our eleven developers. Distributing Nipype knowledge from two experts to nine novices over two days was not an easy beginning task, but most of the novices had gained knowledge of Nipype usage by the end of the hackathon. Some work began during the hackathon converting HCP scripts into Nipype pipelines, however not much progress was made due to the unanticipated large scope of work. The second day, an epiphany came about that the original goal, as stated, would have only involved making five top-level wrappers for the five HCP top-level scripts. This also slowed some progress. The secondary goal of generalizing the HCP scripts was discussed, but not thoroughly explored or documented. There has only been some progress in generalization I am aware of in the Neuroimaging Lab (PI: Damien Fair, PA-C, PhD), at OHSU. This turnout of developers during an open hackathon is encouraging and demonstrates the importance of trying to fuse these two systems (Nipype and the HCP scripts) to work together. Work on the repository halted after the hackathon, but the team is still available.

Conclusions

More work is needed to truly contribute back to the HCP Pipelines https://github.com/Washington-University/Pipelines. The greatest achievement of the hackathon project was forming a collaborative team of interested Nipype developers who were trained and are ready to continue collaborating across seven institutions. Future work will continue trying to achieve the original goals as stated, but may need an organizer to hold the team accountable to deadlines. To get involved with this project, please contact Eric Earl, earl@ohsu.edu.

Availability of supporting data

More information about this project can be found at: https://github.com/ericearl/hcp2nipype-hack2015/

Competing interests

None.

Author’s contributions

EE wrote the report, EE and all other authors wrote the software.

Acknowledgements

Report from 2015 OHBM Hackathon (HI). The authors would like to thank the organizers and attendees of the 2015 OHBM Hackathon.

References

1. Glasser MF. The minimal preprocessing pipelines for the Human Connectome Project. Neuroimage. 2013; 80: 683–691.