https://doi.org/10.2352/ISSN.2470-1173.2021.14.CVAA-014

A Web-based Visualization Tool for Multispectral Images

Snehal A. Padhye, David Messinger, James A. Ferwerda; Chester F. Carlson Center for Imaging Science, Rochester Institute of Technology; Rochester, New York

Abstract

Multispectral imaging has been a valuable technique for discovering hidden texts in manuscripts, learning the provenance of antique books, and generally studying cultural heritage objects. Standard software used in displaying and analyzing such multispectral images are often complex and requires installation and maintenance of custom packages and libraries. We present an easy-to-use web-based multispectral imaging visualization tool that enables simultaneous interaction with the information captured in different spectral bands.

Introduction

Multispectral Imaging has led to significant discoveries in historical documents and artworks. It has been used for better interpretation of ancient manuscripts and paintings [1]. It was used in extracting undertexts from Archimedes palimpsests [2]. Multispectral Imaging has also been used in identifying stains and discoloration on old documents by measuring their spectral signatures [3]. In multispectral imaging, each spectral band provides unique information which is used in overall analysis of the cultural heritage document. It is important to accurately capture this data at multiple wavelengths, and it is equally important to have the means to visualize these data in a way that brings out a meaningful interpretation.

Standard software packages used for visualization and analysis of multispectral data often require local installation of custom applications and libraries, leading to system security problems. They are also platform dependent and require a user to keep all packages updated for best performance and compatibility of the software. In contrast, we have developed a tool for multispectral data visualization through a resource readily available to modern computer users – web browsers! We use computer graphics to process and display multispectral data in our visualization tool. Web-based graphics capabilities have evolved dramatically in last few years, and with the introduction of WebGL [4], high-quality, real-time graphics can be easily rendered by most browsers. We use Three.js [5] – a library and API for WebGL, to create an interface for interactive multispectral visualization. Users can access and interact with their multispectral data just by putting its URL into a browser on any web-compatible device.

Previous Work

There are many software packages for multispectral image visualization, analysis, and processing. ENVI [6] is one of the most popular packages for multispectral image processing and visualization in remote sensing and document imaging. It also supports multiple windows, displaying data from different spectral bands in each session. MATLAB [7] is another immensely popular package with multispectral imaging support. The ImageLab package [8] also supports multispectral image analysis. While all these packages provide tools for multispectral data visualization and analysis, their interfaces can be cumbersome and unintuitive to use, and they require local installation of their custom apps, libraries, and drivers. Also, all of them are paid software.

In addition to commercially available software, researchers have also developed multispectral imaging tools that are freely available. Opticks [9] is one such freeware package. MultiSpec [10] is another easy-to-use multispectral data analysis package which is largely designed for remote sensing applications. Gerbil [11] is yet another open-source interactive multispectral data visualization software. It also supports visualization of topological features along with the spectral properties. While all these packages are of great value, they still require installation and maintenance of software components which can create security issues and often have steep learning curves which can limit widespread use.

The introduction of WebGL has led to many web-based 3D publishing and visualization platforms [12]. Sketchfab [13] is the most popular commercial platform used to display and interact with user created 3D models. Developed for a broader audience, it lacks flexibility to support domain specific requirements. Smithsonian Museum X3D [14] is another tool used to visualize 3D artefacts. It is more flexible than Sketchfab and it was developed to encompass the huge variety of cultural heritage objects in the Smithsonian museum. It is owned by Autodesk and is restricted for wider use. 3DHOP [15] is an opensource 3D cultural heritage model visualization platform built on WebGL. It has rich set of tools for interaction with the 3D models.

All these platforms are excellent for 3D model interaction and visualization and they may also support integration with different media. However, there is a disconnect between these 3D tools and those focused on multispectral data. Since, both 3D and multispectral data have been of great importance in the cultural heritage domain, we have developed a tool that supports both these aspects. In the following section, we describe this web-based, interactive, 3D multispectral data visualization tool that addresses these issues, and provides simple, secure, and widespread access to multispectral image data.

Web-based Multispectral Visualization

Instead of displaying a traditional two-dimensional image for visualization of multispectral images, we employ 3D computer graphics to render a scene to depict the spectral data. The development of the tool is driven by following objectives:

- Multispectral data visualization
- Simultaneous display of multiple bands
- Browser based
- Intuitive and easy to use interface
- No applications or libraries to install and maintain
- No security issues
- Free and open source
- Support for richer datasets (shape, texture, material)

To support the kind of features we are aiming for our tool, we require a platform independent framework that supports 3D

representation. WebGL, based on OpenGL ES 2, has brought the power of complex 3D graphics to browser and every modern browser supports it. We found the idea of building a browser-based interface very attractive since browsers protect users from security threats arising from installing third-party software and enables users to access the tool using standard URL links.

We can represent our application [16] in three layers. The lowest layer includes graphics enabling libraries, in our case it is WebGL. The middle layer builds over the lowest layer libraries to provide higher level functionalities. Our tool uses Three.js as the middle layer library. Finally, the last layer contains the custom application-specific code. The overall interface code is written in HTML and JavaScript. The layers along with the user interaction completes our WebGL-based application ecosystem as shown in Figure 1.

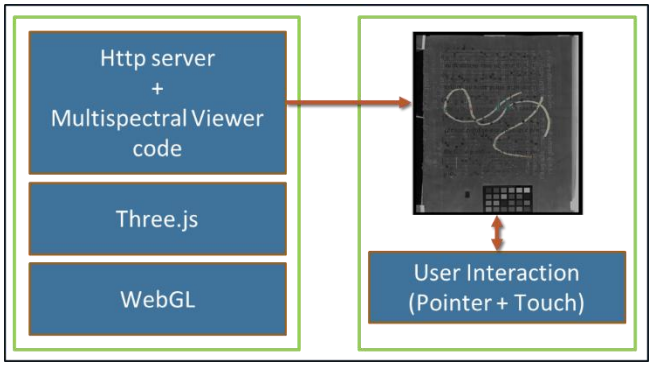


Figure 1. *Application Ecosystem: WebGL forms the lowermost layer providing the basic graphics capability. Three.js uses WebGL to provide higher level functionalities to manage a rendered scene as a middle layer. The topmost layer consists of our application code, built over Three.js, to provide multispectral visualization. The application can be run through an HTTP server to display multiple spectral bands of an object. The user can then observe and interact with the rendered object.*

A typical Three.js pipeline is shown in Figure 2. An object is characterized by its geometry and material properties. Sample can be planar, spherical, or described parametrically, and materials can vary in their diffuse and specular reflectance and transmittance properties. These properties together form a Mesh that is added to a Three.js Scene. The scene also requires an illumination source and a camera to provide the lighting and viewing angles. Finally, the Scene is rendered using the WebGL renderer and displayed in the browser.

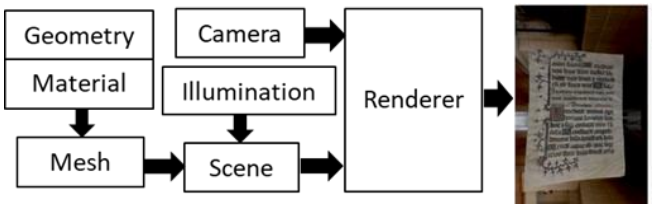


Figure 2. *Three.js Pipeline: The geometry and material properties of an object are specified and turned into a mesh that is a component in a scene that includes lights and a camera. The renderer takes this information and produces an image. Using Three.js all these steps can be performed in real-time on a standard web-browser.*

The use of 3D computer graphics provides the flexibility of adding topographical, reflectance, translucency, and other surface features to the visualization of the object. We represent these properties in a form of 2D maps with each pixel location denoting the property value. To use these maps, physically located on the system, we ask our browser to fetch the files for our rendering. For security, browsers follow a standard for communication between different webservers in which it provides the origin of the file request in its header. Cross Origin Resource sharing (CORS) is the mechanism that grants access to the resources across origins. When we try to run our HTML interface directly, without a webserver, the CORS essentially identifies the origin as Null and does not permit us to access our image maps. To overcome this problem, we can either do system level changes to convey the CORS to allow resource sharing at our local system or we can simply run a local webserver to mitigate the issue. The server can be a simple Python HTTP server that can be started by running a command 'python -m HTTP. Server 8000' on the system's terminal.

Considering all scenarios, we opted for a client-server configuration for our application. The client is basically the HTML interface code that runs our visualization tool in a browser and the server is any basic webserver that serves the HTML application code. The configuration further provides two possible ways of running the application:

Local Setup

In the local configuration as shown in Figure 3, both server and the client run on a local network setup. An user can type the link to the server (' http://localhost:8000/') on any browser to start the visualization interface. Any other device in the local network can also access this tool by typing the relevant address of the application in their browsers (for e.g. http://<server IP address: port>).

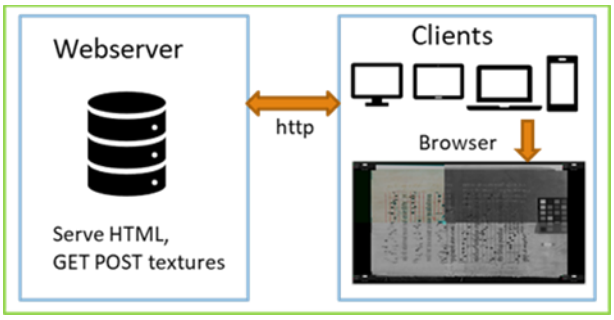


Figure 3. *Local setup: The server and the client run locally (represented by the outer boundary). The client can be any browser-running device. The client enters the server's local address and the port on which it is running, and the application runs on the client through the http protocol. The user can then have seamless interaction with the object through the client system.*

Remote Setup

The server in the local setup when hosted on a machine that is accessible over the Web, enables the user to access and visualize their data remotely. Thus, a user can continue working remotely by accessing the application, through its URL, on any web-connected device as shown in Figure 4.

Multispectral Visualization Tools

Our current implementation provides three visualization/view modes which we call Quad viewer, Multispectral lens, and Multispectral highlighter. Each of the modes require users to provide a folder with the target images as input. The application then takes the first few images in the folder path to initialize the viewer and populates all the available image options in the

dropdown on the GUI. Note that the upload operation does not share the data to the server. It is always local to the user. The application supports static files. The displayed band can be selected from the dropdown and changed on the viewer. The viewers also provide freedom to add any other geometrical and material properties along with the spectral data for a richer representation of the rendered object. Once the server is running, the user is only required to enter the local or remote URL (according to the configuration) on a browser to launch any of the view modes. Figure 5 shows a screenshot of the application launched on a local device through a webserver hosted on GitHub. It shows an example of the remote configuration of the application. The homepage shows all the available modes of visualization and one can scroll through to see each of them.

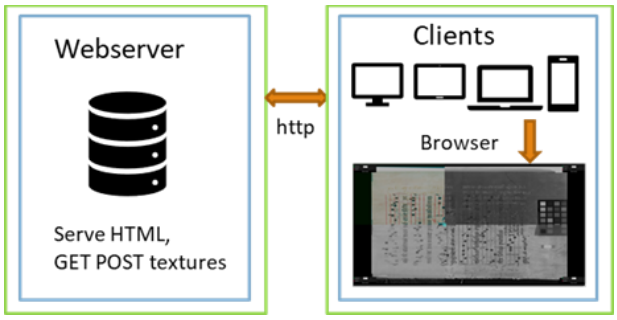


Figure 4. *Remote Setup: The server and the client are on separate systems connected over Web. The client runs the application by connecting to the server through its IP address and its port over http protocol.*

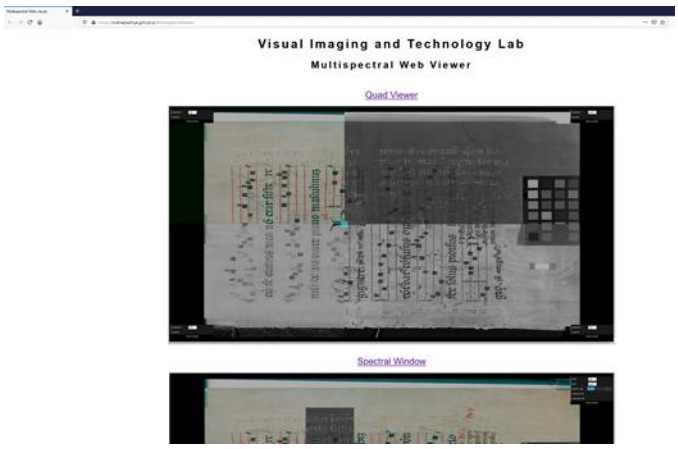


Figure 5*. Application launched on a user device through a remote configuration. The homepage lists all the available options for visualization. The user can click on any of the options to launch the viewer.*

Quad Viewer

A Quad viewer enables simultaneous visualization of four bands of an object captured at multiple wavelengths. Here, we render multiple scenes simultaneously (one per spectral band) in a given viewport or browser screen and the renderer then displays only those spectral bands that are selected by the user. It has four small GUIs at each corner for managing image displayed in each quadrant. It has a cyan-colored knob at the center which enables resizing of each quadrant to dynamically analyze the document at different spectral bands. A snapshot of the Quad viewer is shown in Figure 6.

Multispectral Lens Viewer

While the Quad viewer provides simultaneous visualization of multiple spectral bands through non overlapping quadrants. The Multispectral lens viewer allows users to visualize different spectral bands simultaneously through an overlapping region called as a window or a lens. The GUI on the right corner provides an option to select a rectangular window or a circular lens shape to discover information from different bands through it. The window/lens can be resized and moved around interactively. There are also options to choose spectral bands to be displayed in the window or lens as well as in the outer region of the sample. Snapshots of the Multispectral lens viewer are shown in Figure 7.

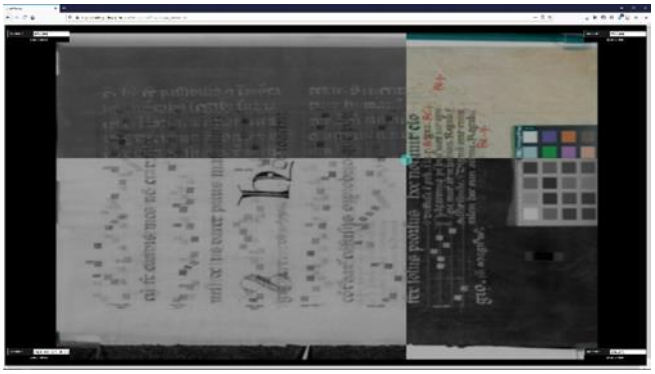


Figure 6. *Web-based multispectral quad image viewer: To access a multispectral dataset, a user types its URL into a browser. The browser then renders the image data and allows the user to interactively split the image into four quadrants and select the spectral bands shown in each quadrant.*

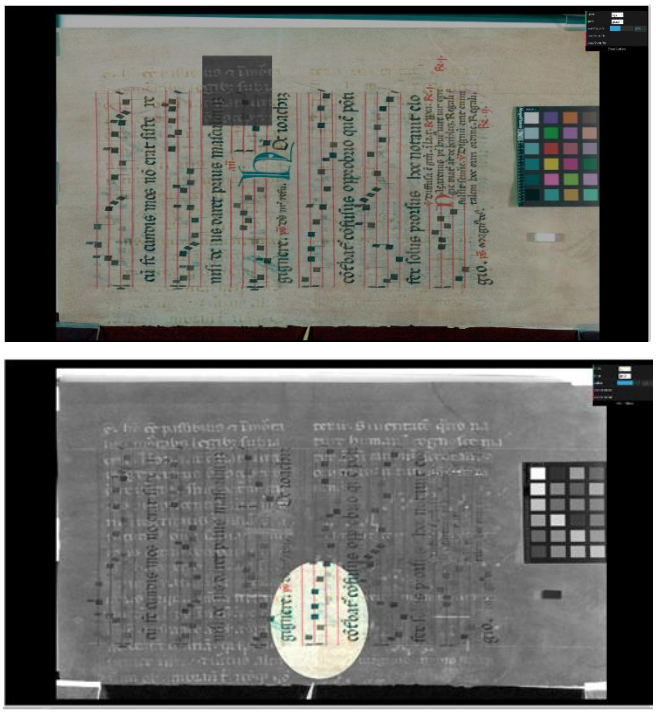


Figure 7. *Multispectral lens viewer: In this viewer, the user can specify the size, position, and spectral properties of a rectangular or a circular region that acts as a 'lens' or filter to reveal particular object properties. This provides increased flexibility in targeted visualization and analysis over the Quad viewer.*

Multispectral Highlighter

Both Quad viewer and Multispectral lens viewer have a well-defined viewing area for managing the spectral bands and the areas seen. The Multispectral highlighter enables a user to visualize a segment as small as a touch or a click on the screen in a different spectral band. The viewing area can be any non-uniform pattern the user wishes to analyze. It consists of a rendered object in a given spectral band and a user can touch, stroke or click and drag over the surface to view that particular portion in a different spectral band. The user can select the spectral band to display in the highlight regions. The application also allows the user to change the width of the strokes at runtime. Figure 8 shows a snapshot of the Multispectral highlighter.

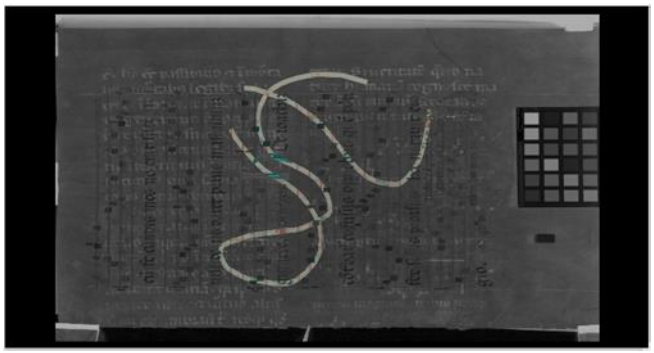


Figure 8. *Multispectral highlighter: Using a mouse or touch screen, user can specify the sizes, positions, and spectral display properties of multiple image regions.*

Assessment

Web-based frameworks have their own limitations. In this section, we discuss each of them and justify their validity for our tool.

Performance

The performance of any web-based visualization application depends on its ability to handle large data files, and its efficiency in rendering the data [17]. Three.js uses the Nexus engine [18] that employs progressive streaming of view-dependent representations along with data compression to streamline transmission. Since, our application does not require very complicated 3D scenes, the only bottleneck should be file size. Very large files (30+ Mb) may slow the rendering initially but will not affect the user interaction once an object is rendered. Any image format supported by Three.js works with the application but PNG (lossless compression) and JPG (lossy compression) are the formats most used. Power of two image sizes are preferable as well since it avoids resampling in the renderer's texturing operations.

Device Support

Device or hardware support is another important property of an application. Three.js use WebGL to render the scene and hence the application can be run on any desktop or mobile browser that supports WebGL. The latest list includes [5] Google Chrome 9+, Firefox 4+, Opera 15+, Internet Explorer 11, Safari 5.1+ and Microsoft Edge.

Security

Security plays a crucial role in installing/using any application. The browser-based approach protects a user from security issues related to installation of third-party software. Browsers operate in a sandboxed environment that makes accessing the local computer resources very difficult. While, this is seen as a disadvantage from a developer's perspective because it makes operations like writing temporary data to the disk very difficult, from a user's perspective, this feature ensures that their system resources are protected and safe.

Provenance and Intellectual Property Protection

Two of the greatest concerns in the cultural heritage community about using web-based visualization and dissemination tools are data provenance and intellectual property (IP) protection. Our application framework provides support for both of these concerns. In both the local and remote setups, data is uploaded and used at the client side. The application just hosts static files that are not stored anywhere other than temporarily in user's browser. This protects the data from being exposed on a network, the user must ensure that they have the data they want to analyze on the client system they are using.

Cost and Accessibility

Cost and accessibility are also important factors in using a visualization tool. Our tool is completely open source and free. We can also add features to customize for a specific application domain. For example, the current version supports data from client side for IP protection, but for open-source projects, the data can be stored and accessed from a server so that every user can visualize the data without having a local copy. The code is available freely on GitHub and can be used by accessing our server at - https://github.com/snehalpadhye/MultispectralViewer.

Limitations and Future Work

Our tools are currently limited to visualizing planar objects such that the geometry and material maps are rendered on a planar surface mesh and the maps can only be in formats that are supported by WebGL and Three.js. We plan to extend our tools to support non-planar objects. We also plan to add ability to rotate the objects in 3D to enhance the rendering of texture and material features along with simultaneous visualization of spectral bands. If required, we also plan to add an option for server-side storage for open-source projects and add image processing features as required by domain-specific applications.

Conclusion

We have presented a set of browser-based visualization tools for multispectral image data. They are free and easy to use for a quick visualization and analysis of multi modal data (geometry, material, multispectral color) without the problems of installing software, or concerns about storage and security issues. The tools are written in HTML and JavaScript and can run on modern desktop, laptop, and mobile devices with just a URL.

Web-based tools for visualization of multispectral data provide powerful, simple-to-use, and secure means to for analyzing and understanding cultural heritage objects. Our hope is that these tools will allow multispectral imaging to provide greater insights into the objects under study.

References

[1] F. Imai, M. Rosen, and R. Berns, “Multi-spectral Imaging of a van Gogh's Self-portrait at the National Gallery of Art Washington DC,” in Image Processing, Image Quality, Image Capture, Systems Conference, Society for Imaging Science and Technology, 2001.

[2] R. Easton Jr, K. Knox., and W Christens-Barry, “Multispectral imaging of the Archimedes palimpsest,” in Proceedings of 32nd Applied Imagery Pattern Recognition Workshop, IEEE-AIPR, 2003.

[3] A. Campagnolo, E. Connelly, and H. Wacha, “Labeculæ Vivæ: Building a Reference Library of Stains for Medieval and Early Modern Manuscripts,” Manuscript Studies: A Journal of the Schoenberg Institute for Manuscript Studies 4(2), 401-416, 2019.

[4] WebGL API, https://www.khronos.org/webgl/.

[5] Three.js API, https://threejs.org/.

[6] ENVI Software, https://www.l3harrisgeospatial.com/Software-Technology/ENVI.

[7] MATLAB Software, https://www.mathworks.com/products/matlab.html.

[8] ImageLab Software, http://www.imagelab.at/.

[9] Opticks Software, https://opticks.org/.

[10] L. Biehl, and D. Landgrebe, “MultiSpec: a tool for multispectral--hyperspectral image data analysis,” Computers & Geosciences. 28. 1153-1159. 10.1016/S0098-3004(02)00033-X, 2002.

[11] J. Jordan, and E. Angelopoulou, “Gerbil - A Novel Software Framework for Visualization and Analysis in the Multispectral Domain, “in Proceedings Vision, Modeling, and Visualization, The Eurographics Association, 2010.

[12] R. Scopigno, M. Callieri, M. Dellepiane, F. Ponchio, and M. Potenziani, “Delivering and using 3D models on the Web: Are we ready?,” in Virtual Archaeology Review, 2017.

[13] Sketchfab, https://sketchfab.com/.

[14] Smithsonian 3D Digitization, https://3d.si.edu/.

[15] M. Potenziani, M. Callieri, M. Dellepiane, M. Corsini, F. Ponchio, and R. Scopigno, “3DHOP: 3D Heritage Online Presenter,” Computers & Graphics,Volume 52, 2015.

[16] M. Potenziani, M. Callieri, M. Dellepiane, and R. Scopigno, "Publishing and Consuming 3D Content on the Web: A Survey", Foundations and Trends in Computer Graphics and Vision, 2018.

[17] Boutsi, Ioannidis, Soile,” An Integrated Approach to 3D Web Visualization of Cultural Heritage Heterogeneous Datasets,” Remote Sens, 2019.

[18] F. Ponchio and M. Dellepiane,” Fast decompression for web-based view-dependent 3D rendering,” in Proceedings of the 20th International Conference on 3D Web Technology, 2015.

Author Biography

Snehal A. Padhye is a third year PhD. student in the Chester F. Carlson Center for Imaging Science at RIT. She received a BS in Electronics from RCOEM and MS in Signal Processing from COEP, both from India. Her dissertation research work focuses on designing hardware and software systems for capturing and visualizing realistic digital models of cultural heritage objects.

David Messinger is a professor in the Chester F. Carlson Center for Imaging Science at RIT. He received a BS in Physics from Clarkson University and Ph.D. in Physics from Rensselaer Polytechnic Institute. His research interests include developing methods to extract quantitative information from spectral imagery, use of physics-based signatures to augment methods of hyperspectral image exploitation and the use of remote sensing techniques for multi-disciplinary research such as archeology, disaster management, and analysis of cultural heritage artifacts.

James A. Ferwerda is an Associate Professor in the Chester F. Carlson Center for Imaging Science at RIT. He received a B.A. in Psychology, M.S. in Computer Graphics, and a Ph.D. in Experimental Psychology, all from Cornell University. The focus of his research is on building computational models of human vision from psychophysical experiments and developing advanced imaging systems based on these models.

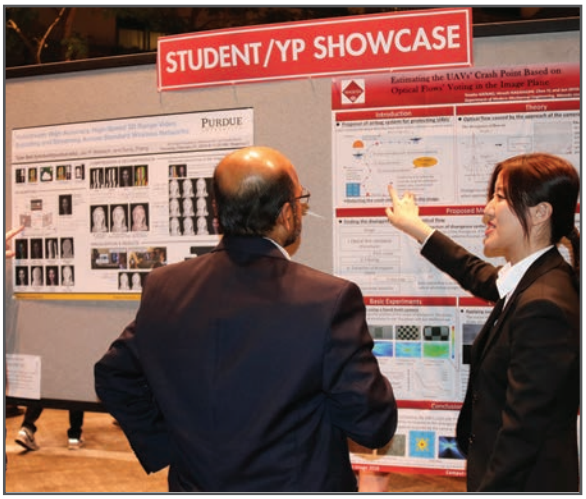
JOIN US AT THE NEXT EI!

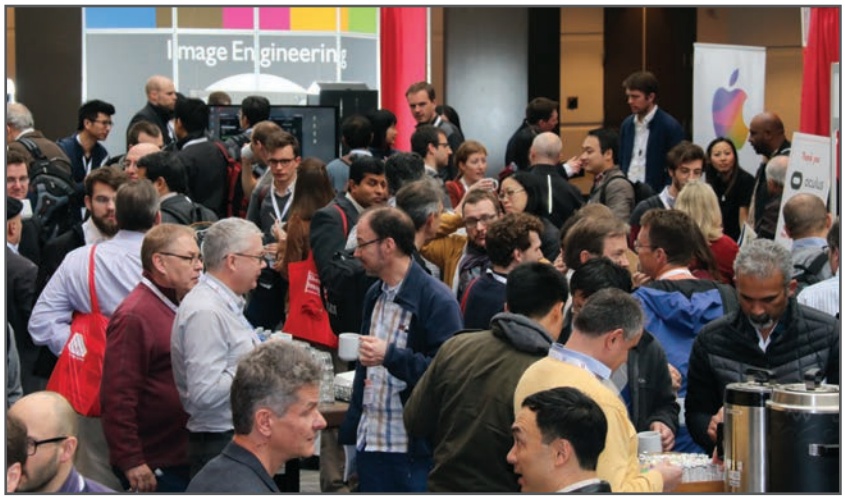
IS&T International Symposium on

Electronic Imaging

SCIENCE AND TECHNOLOGY

Imaging across applications . . . Where industry and academia meet!





• SHORT COURSES • EXHIBITS • DEMONSTRATION SESSION • PLENARY TALKS •
• INTERACTIVE PAPER SESSION • SPECIAL EVENTS • TECHNICAL SESSIONS •

www.electronicimaging.org

IS&T
imaging.org