Image Systems Simulation for 360° Camera Rigs

Trisha Lian, Joyce Farrell, Brian Wandell
Stanford Center for Image Systems Engineering, Stanford, CA

Abstract

Camera arrays are used to acquire the 360° surround video data presented on 3D immersive displays. The design of these arrays involves a large number of decisions ranging from the placement and orientation of the cameras to the choice of lenses and sensors. We implemented an open-source software environment (iset360) to support engineers designing and evaluating camera arrays for virtual and augmented reality applications. The software uses physically based ray tracing to simulate a 3D virtual spectral scene and traces these rays through multi-element spherical lenses to calculate the irradiance at the imaging sensor. The software then simulates imaging sensors to predict the captured images. The sensor data can be processed to produce the stereo and monoscopic 360° panoramas commonly used in virtual reality applications. By simulating the entire capture pipeline, we can visualize how changes in the system components influence the system performance. We demonstrate the use of the software by simulating a variety of different camera rigs, including the Facebook Surround360, the GoPro Odyssey, the GoPro Omni, and the Samsung Gear 360.

Introduction

Head mounted visual displays can provide a compelling and immersive experience of a three-dimensional scene. Because the experience can be very impactful, there is a great deal of interest in developing applications ranging from clinical medicine, behavioral change, entertainment, education and experience-sharing [1] [2].

In some applications, computer graphics is used to generate content, providing a realistic, but not real, experience (e.g., video games). In other applications, the content is acquired from a real event (e.g., sports, concerts, news, or family gathering) using camera arrays (rigs) and subsequent extensive image processing that capture and render the environment (Figure 1).

The design of these rigs involves many different engineering decisions, including the selection of lenses, sensors, and camera positions. In addition to the rig, there are many choices of how to store and process the acquired content. For example, data from multiple cameras are often transformed into a stereo pair of 360° panoramas [3] by stitching together images captured by multiple cameras. Based on the user's head position and orientation, data are extracted from the panorama and rendered on a head mounted display. There is no single quality-limiting element of this system, and moreover, interactions between the hardware and software design choices limit how well metrics of individual components predict overall system quality. To create a good experience, we must be able to assess the combination of hardware and software components that comprise the entire system.

Building and testing a complete rig is costly and slow; hence, it can be useful to obtain guidance about design choices by using a simulation of the system. This paper describes software tools that simulate controlled 3D realistic scenes and image acquisition systems, in order to generate images produced by specific hardware choices. These images are the inputs to the stitching and rendering algorithms. The simulation enables engineers to explore the impact of different design choices on the entire imaging system, including realistic scenes, hardware components, and post-processing algorithms.

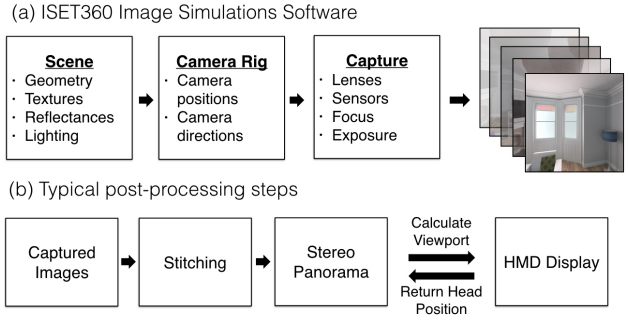

Figure 1. *Overview of the hardware and software components that combine in an camera rig for an immersive head-mounted display application. (A) The simulation includes a 3D spectral scene, the camera rig definition, and the individual camera specifications. This simulation produces a set of image outputs. (B) The images are then processed by a series of software algorithms. In this case, we show a pipeline that produces an intermediate panorama representation and the viewport calculations that render an image dependent on the users head position.*

Software Implementation

The iset360 software, which models the image capture pipeline of 360 camera rigs, has portions in MATLAB and portions in C++. The simulation software is freely available in three repositories within the ISET GitHub project: *https://github.com/ISET* [1].

Figure 2 and Figure 3 summarize the initial stages of the workflow. The first portion of the code creates realistic 3D scenes and calculates the expected sensor irradiance given a lens description. To do so, we start with a 3D, virtual scene that is constructed using 3D modeling software (e.g. Blender or Maya). The scene is converted into a format compatible with PBRT [4], which is implemented in C++.

PBRT is a quantitative computer graphics tool that we use to calculate the irradiance at the sensor as light travels from the 3D scene, through the lens, and onto the sensor surface. We augmented the PBRT code to return multispectral images, model lens diffraction and simulate light fields [5]. To promote platform in-

[1] The three repositories are iset360, iset3d, and isetcam

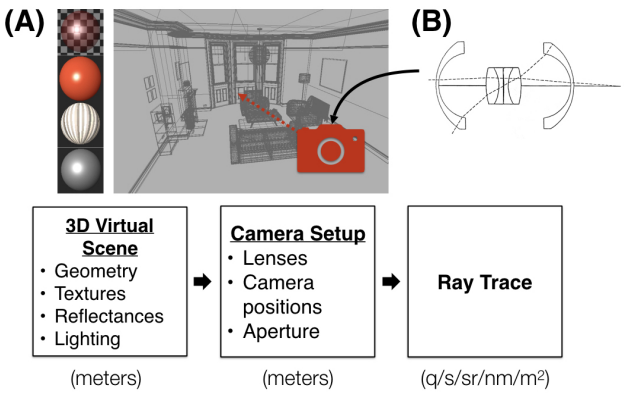

Figure 2. *The iset360 software uses computer graphics to define a 3D scene spectral radiance and calculate how that passes through the lens to become the sensor spectral irradiance. (A) The scene geometry is shown as a set of meshes. The camera position and lookAt direction are indicated by the red icon. Example object textures are shown in the panel at the left. (B) The scene radiance is traced through a multi-element spherical lens to form the sensor irradiance. The lens file describe the surface positions, curvatures, apertures and wavelength-dependent indices of refraction. A wide-angle lens prescription is shown above.*

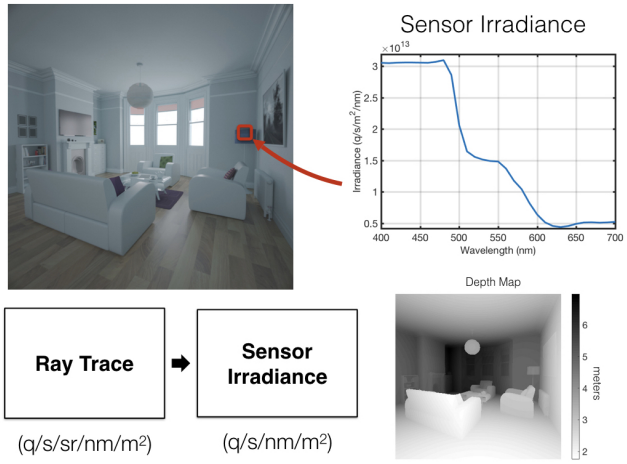

Figure 3. *The second stage of the software uses a ray-tracer to compute the sensor irradiance which is represented here by an image. The sensor irradiance is multispectral; above we show the irradiance curve for a single pixel, as well as the depth map generated by the renderer.*

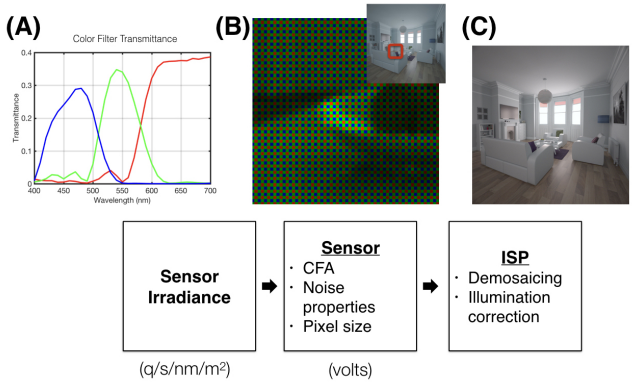

Figure 4. *The final stage of our software models the sensor and ISP properties of the simulated camera. (A) The color filter transmittance is shown for the simulated scene. (B) The raw sensor data overlaid with the Bayer filter pattern for a small portion of the sensor is shown. (C) The final image (rgb) after image processing. These images can eventually be passed to a post-processing pipeline, as shown in Figure 1.*

dependent sharing, the augmented PBRT code is compiled into a machine-independent Docker container [2]. This portion of the code is in the iset3d repository.

A small library of pre-converted and formatted scenes from [6] are included with the software. The scene files describe the size and distance of objects in meters, material properties such as image textures and BRDFs, and the placement and type of lights. In addition, one can specify the spectral power distribution of the lights, as well as the spectral reflectance of objects. Many of these values can be changed programmatically once the scene has been imported into the iset360 software.

In the expected usage, the user controls the iset360 simulation pipeline using a Matlab script. Typically, the user imports the PBRT scene data using the command *recipe = piRead(sceneFile)*. The return is a Matlab object (*recipe*) whose parameters specify the scene data and camera settings. We loop through each camera in the rig, setting its position, lens and sensor parameters, before calculating its sensor irradiance (*irradiance = piRender(recipe)*, units: photons/s/nm/m).

Figure 4 summarizes the final portion of the code that converts the sensor irradiance into the expected image (rgb) data. The format of the sensor irradiance data is compatible with isetcam, Matlab code which models the geometric, colorimetric, and electrical properties of the pixels and sensor [7]. A large number of parameters can be specified by adjusting the iset360 recipe and the isetcam sensor parameters. These range from scene properties (e.g., lighting, materials, geometry), to camera array (positions, lens prescriptions), to sensor properties (e.g., pixel size, color filter array, sensor and pixel noise, etc.)

For example, we specify a camera position and viewing direction using a LookAt parameterization which describes an origin (from), a target (to) and an up direction (up). We describe multi-element lenses by their component positions, curvatures, thicknesses, diameters, and wavelength-dependent indices of refraction. The toolbox includes a small library of spherical lenses, and additional ones can be added when the lens prescription is known. Within the iset360 camera structure, we specify the aperture diameter and the position of the sensor. Within the isetcam structures we specify the color filter array, pixel sizes, and electrical noise. The iset360 repository includes software that creates the images included here and shows how the functions are called and parameters are set.

There is one additional feature of the simulation environment that may prove important. It is possible to create ground-truth pixel-level representations of the depth map as well as the perfect viewport image. Ultimately, such information may provide an opportunity to improve stitching and viewport rendering algorithms.

Computation time is largely consumed by the ray-tracing

[2] Available at: *https://hub.docker.com/r/vistalab/pbrt-v3-spectral/*

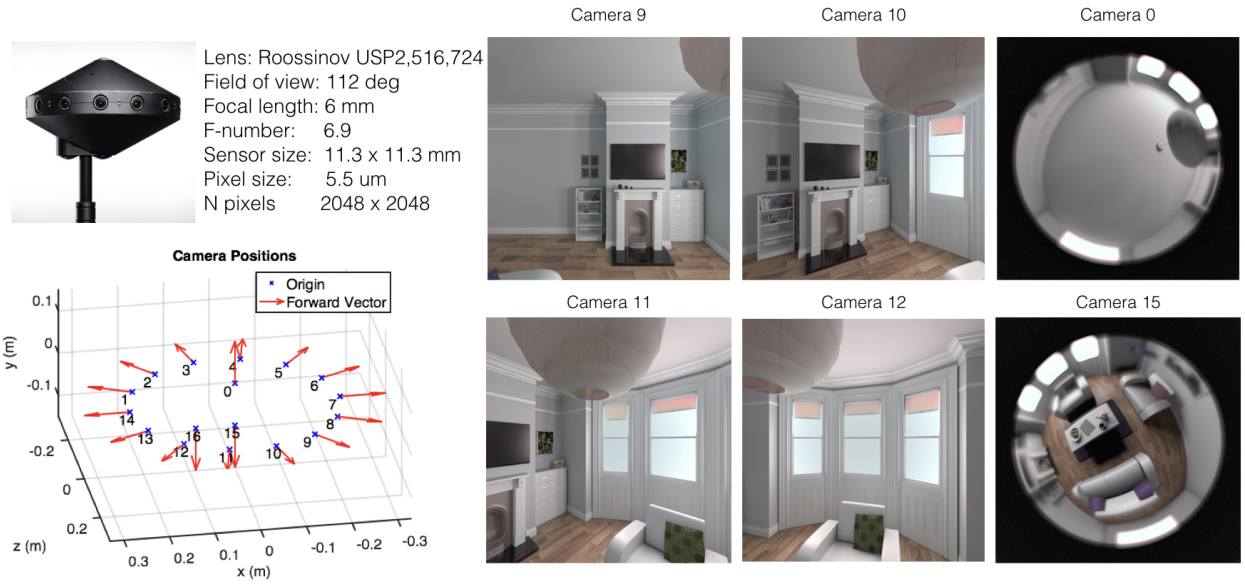

Figure 5. *Simulation of the Facebook Surround360 camera rig. (A) An image of the rig and list of simulation parameters for the lens and sensor. (B) The camera positions (blue dots) and lookAt directions (red). Cameras 1-14 are positioned around the circumference. Cameras 0, 15 and 16 are pointing toward the ceiling and floor. (C) Images from four cameras on the circumference (9,10,11,2) and two fisheye cameras (0,15).*

step; the other computations take less than a minute, but it takes about 2 hours to render a 4 megapixel image on an 8-core machine. The rendering time for the whole rig can be minimized using parallel machines for each camera. To simplify this, we made the computation compatible with Google Cloud and implemented a Matlab toolbox[3] to set up the connection to Google Cloud. Once the toolbox is initiated with a user's account, the render command can be invoked so that each camera image is rendered in parallel on the cloud, in its own docker container, as managed by kubernetes (k8s). Each image renders in about an hour, but there is little penalty for multiple images. For example, a set of sixteen 4-megapixel images from a camera rig can be rendered in parallel in less than an hour, corresponding to about 4 minutes per image to simulate the rig.

Results

We created iset360 scripts to model four different camera rigs. We have access to some but not all of the parameters. To create the simulations we used the available information, and when we could not find an exact part we used components with similar properties. For example, we chose lens prescriptions from [8] that matched the field of view of each camera. We also simulated the correct sensor resolution and size and approximated the camera positions. Key rig parameters are listed in Figures 5-8; the approximations can become more exact as we obtain more vendor information.

The scripts included in the iset360 github repository specify the full set of parameters needed to generate raw sensor data. To simplify the visual presentation of the images, we converted the raw sensor data to sRGB format using the default image processing settings in the isetcam toolbox.

[3] *https://github.com/ISET/isetcloud*

Facebook Surround360

Facebook has published the specifications of the Surround360 rig along with software that implements the stitching algorithm [9]. The rig has two sets of cameras: The first set comprises 14 cameras placed in a circle around a 460 mm diameter circle. Each camera has a wide-angle lens. The second set comprises three cameras with fisheye lenses placed on the top and bottom of the rig (Figure 5).

We simulated the first set of cameras using a wide angle lens with a FOV of 112° and a focal length of 6 mm to best match the actual wide-angle lens on the Surround360 (Sunex DSL318, FOV = 110°, Focal length = 7.0 mm). Due to limitations on the wide-angle lens prescriptions available to us, we used a relative aperture of (f/8) which differs from the Sunex (f/2.4). We simulated a 4.1 megapixel, 1" sensor with 5.5 um pixel size to match the Point Grey Grasshopper 3 used in the Facebook Surround360.

In this configuration, there is substantial overlap in the field of view between adjacent cameras. In principle, it is possible to use such images as inputs to stereo algorithms and make depth estimates of much the surrounding environment. The image pairs are unlike many stereo rigs or the human visual system in which the two eyes converge on a common point in the scene; in this rig the LookAt directions of adjacent cameras diverge.

Furthermore, notice that the images in Cameras 9 and 10 - as well many other examples - include objects with very different intensities. Compensating for the difference in level and ambient illuminant color will be an important part of panorama construction. Also, the unequal spatial resolution of the two fisheye cameras are a potential source of difficulty in achieving a uniform spatial resolution as the viewer looks around.

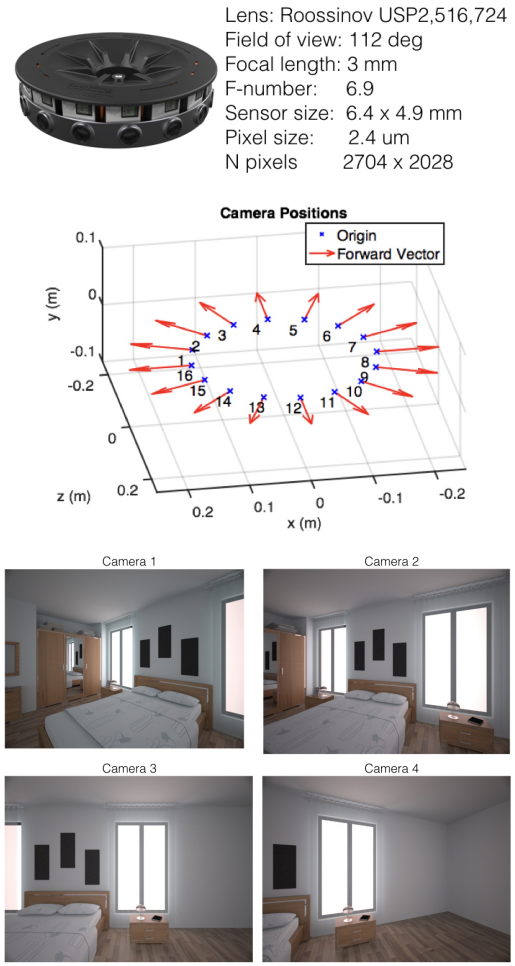

Figure 6. *Simulated images from the GoPro Odyssey rig. This rig consists of 16 GoPro cameras arranged in a ring. Four images from adjacent cameras are shown on the right. Other details as in Figure 5.*

GoPro Odyssey

The GoPro Odyssey rig (Figure 6) consists of 16 cameras (GoPro Hero) arranged in a ring with a diameter of roughly 300 mm, smaller than the Facebook Surround360. The GoPro Hero sensor size is smaller than that of the Facebook sensor. To account for this difference, we scaled the same 112° FOV wide-angle lens used in the Surround360, to produce an image that matched the smaller sensor size. Due to the extra cameras and smaller rig radius, the images from the Odyssey rig overlap more than the images captured on the Facebook rig, which may make stitching more robust. The absence of the fisheye lens cameras limits the ability to provide a good view when looking directly up or down.

Samsung Gear 360

The Samsung Gear 360 design (Figure 7) is very different from the rings shown previously. This rig includes only two cameras with fisheye lenses placed back to back. The camera field of views do not overlap, and thus the stitching algorithm incorporated with the camera produces monoscopic panoramas, not stereoscopic ones. The dual lenses on the Gear 360 have a FOV of

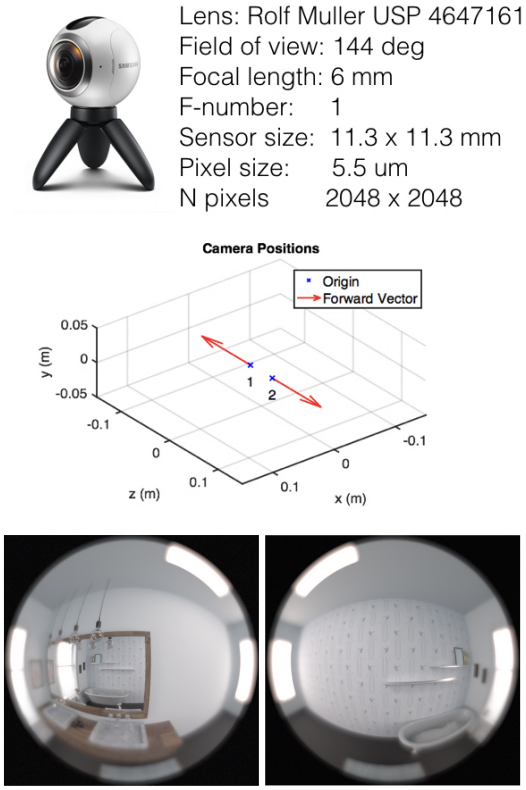

Figure 7. *Simulated images from the Samsung Gear 360, which consists of two camera with fish-eye lenses facing opposite directions.*

180°, however the fisheye lens prescriptions available to us have a maximum FOV of 144°, so the simulation is missing portion of the image. Even with the full 180° the stitching algorithms for this camera must compensate for the large geometric distortions and inhomogeneous spatial resolution of the two fisheye lens images.

GoPro Omni

The GoPro Omni (Figure 8) has six outward facing cameras placed on the surface of a cube. Like the Samsung Gear 360, this rig is used to produce monoscopic panoramas. The simulation uses a wide-angle lens with 112° FOV to match the sensor size, while the actual GoPro camera has a slightly larger FOV of 122.6°. Hence, the images in our simulation somewhat underestimate the overlap between images from adjacent cameras.

Discussion

The iset360 simulation images can be used in several different ways. In one application, the user can evaluate how variations in camera position that might arise during manufacturing would affect the stitching algorithms performance. Similarly, the user can evaluate how different lens choices affect the amount of data available for the pipeline to generate viewports at challenging positions, such as the poles.

Iset3d includes methods that generate pixel-level specifications of the scene properties, such as the distance to each point in the image. In addition, the simulation contains information on

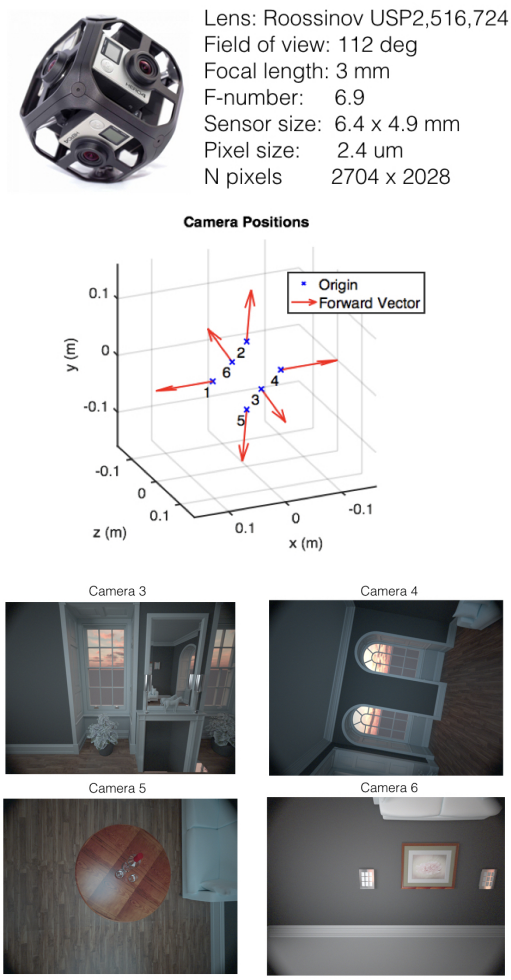

Figure 8. *Simulated images from the GoPro Omni rig which consists of 6 cameras arranged in a cube.*

object locations, material properties and lighting. Such precise information cannot be easily obtained when testing real rigs. This information can support the at least two types of algorithm development.

First, one can generate a perfect panorama directly from the rays in the scene and compare it to a panorama generated by stitching together images captured by a rig to see how accurately the scene was reproduced. This information can be used to evaluate the overall image quality and accuracy of the system. Second, the raw sensor data and ground truth data can be used with machine-learning methods, to derive stitching algorithms that produce accurate stereo panoramas. For example, one can simulate an ideal camera rig and produce a target stereo panorama. Then one can train a network to learn the transformation from the rig under test to the ideal representation. Finally, the iset360 methods permit the user to simulate the capture of exactly the same test scene with different rigs. Comparisons of two rigs with exactly the same scene is difficult to achieve with real measurements. The simulation method enables the user to specify image quality and test targets and then to use the simulation to evaluate two different rigs from the same conditions.

Author Biographies

Trisha Lian received her BS in Biomedical Engineering from Duke University (2014) and is currently a PhD student in Electrical Engineering at Stanford University. Her work has focused on the development of simulation tools for novel camera systems, as well as simulation of the human visual system.

Joyce Farrell is the Executive Director of the Stanford Center for Image Systems Engineering and a Senior Research Associate in the Department of Electrical Engineering at Stanford University. She received a doctorate from Stanford in 1981 and has worked at several companies and research institutions, including the NASA Ames Research Center, New York University, the Xerox Palo Alto Research Center and Shutterfly.

Brian A. Wandell is the first Isaac and Madeline Stein Family Professor. He joined the Stanford Psychology faculty in 1979 and is a member, by courtesy, of Electrical Engineering, Ophthalmology, and the Graduate School of Education. He is Director of Stanfords Center for Cognitive and Neurobiological Imaging and Deputy Director of Stanfords Neuroscience Institute. Wandells research centers on vision science, spanning topics from visual disorders, reading development in children, to digital imaging devices and algorithms for both magnetic resonance imaging and digital imaging.

References

[1] L. M. Williams, A. Pines, A. N. Goldstein-Piekarski, L. G. Rosas, M. Kullar, M. D. Sacchet, O. Gevaert, J. Bailenson, P. W. Lavori, P. Dagum *et al.*, "The engage study: Integrating neuroimaging, virtual reality and smartphone sensing to understand self-regulation for managing depression and obesity in a precision medicine model," *Behaviour research and therapy*, 2017.

[2] J. Bailenson, *Experience on demand : what virtual reality is, how it works, and what it can do*. New York: W.W. Norton & Company, 2018.

[3] S. Peleg, M. Ben-Ezra, and Y. Pritch, "Omnistereo: Panoramic stereo imaging," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 3, pp. 279–290, 2001.

[4] M. Pharr, W. Jakob, and G. Humphreys, *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016.

[5] A. Lin, J. E. Farrell, and B. A. Wandell, "Spectral optics simulation for rapid image systems prototyping: Ray-tracing, diffraction and chromatic aberration," in *Applied Industrial Optics: Spectroscopy, Imaging and Metrology*. Optical Society of America, 2014, pp. JW3A–2.

[6] B. Bitterli, "Rendering resources," 2016, https://benedikt-bitterli.me/resources/.

[7] J. Farrell, G. Ng, X. Ding, K. Larson, and B. Wandell, "A display simulation toolbox for image quality evaluation," *Journal of Display Technology*, vol. 4, no. 2, pp. 262–270, 2008.

[8] W. J. Smith, *Modern Lens Design (McGraw-Hill Professional Engineering)*. McGraw-Hill Education, 2004.

[9] B. Cabral, "Introducing facebook surround 360: An open, high-quality 3d-360 video capture system," 2016, https://code.facebook.com/posts/1755691291326688/introducing-facebook-surround-360-an-open-high-quality-3d-360-video-capture-system/.