FAR: Flooring in Augmented Reality

Ryan K. West

Brigham Young University

December 9, 2021

Abstract

Flooring in Augmented Reality (FAR) is an Android app which improves upon the state of the art in floor covering visualization by increasing the ease of use and immersiveness for floor covering augmented reality apps and providing real-time detection and occlusion of flooring surfaces. It achieves this goal by leveraging AR Foundation's Plane Detector for locating the flooring plane and using real-time semantic segmentation with a DeepLabV3 network to generate a mask of the floor. This mask is further refined through watershed segmentation and then projected back onto the flooring plane to clip it to the real floor's extents. For enhanced realism, lighting conditions are estimated by masking, blurring, and normalizing the captured image of the floor. While FAR is still in the prototype phase, it succeeds as a proof of concept for these augmented reality techniques and serves as a strong foundation for future AR flooring development.

Keywords: AR, augmented reality, semantic segmentation, DeepLab, watershed segmentation, lighting extraction, texture projection

FAR: Flooring in Augmented Reality

The best businesses in the hospitality industry don’t just cater to their guests, they provide an unforgettable experience that begins the moment they arrive. Every detail of the venue – from floor to ceiling – contributes to this carefully crafted aesthetic and must, therefore, be planned meticulously by interior designers. Throughout the years, designers have turned to increasingly realistic visualizations in order to better understand how guests will experience the spaces they visit. Now, advances in virtual reality technology have brought an unprecedented level of realism to interior designers’ visualization process, allowing them to walk through virtual spaces and experience the atmosphere for themselves.

In situations where a CAD model of the venue to be built is available, virtual reality provides an excellent medium for visualization. However, in situations where there is no CAD model available, such as when an old building is renovated, it is prohibitively expensive to construct one for a virtual reality visualization. In these situations, interior designers usually fall back on older, less immersive visualization techniques. In order to improve the realism of visualizations for these prebuilt spaces, I present an augmented reality system called Flooring in Augmented Reality (FAR) for interior designers to easily visualize alternative floor coverings in a space. This augmented reality app overcomes the limitations inherent in virtual reality solutions because designers are able to visualize their floor covering changes in the actual physical space instead of in a CAD model. This system improves upon the state of the art in floor covering AR visualization by providing an easier to use, more immersive experience for users than was previously available. By using this augmented reality system, designers will be able to interactively modify the floor covering of their venue and see their changes in real-time as they walk through the physical space.

Related Work

Since the choice of floor covering is so important to a venue's atmosphere, many tools already exist to help interior designers visualize their space with new flooring. Currently available tools vary considerably in their ease of use, quality of visualization, and immersiveness.

One of the leading solutions for floor covering visualization is a white-label web app called Roomvo (Leap Tools Inc, 2021). Roomvo excels at ease of use and visual fidelity, making it a popular choice for brands to license for their own product lines. This web-based tool works by taking a user-uploaded photograph and automatically segmenting the floor from the rest of the image. It then infers a perspective projection for the flooring and floods the space with a carpet pattern. Finally, lighting conditions such as shadows and highlights are extracted from the original image and transferred to the simulated carpet (See Figure 1). While Roomvo is easy to use and produces high quality visualizations, the tool is not very immersive since it takes several seconds to process inputs and only produces a still image instead of a live view.

On the other end of the immersiveness spectrum in the realm of virtual reality, there exist apps such as Planner5D (Planner 5D. Interior Design: Room, Home, Floorplan, 2015). Planner5D allows a user to upload a CAD model of their space, or to recreate the space using their app. Once this is done, the user is free to swap out floor coverings – or any other elements of the design. They can then use the Google Cardboard VR viewer to look around the space to get a feel for the atmosphere they created. Apps like Planner5D excel at providing immense customizability and immersion, however the requirement to rebuild the space from scratch or find a CAD model is prohibitively time consuming and difficult, reducing the app's utility.

A third kind of app for flooring visualization is the augmented reality app. Apps such as AR Home Flooring fall into this category (AR Home Flooring, 2019). In order to use AR Home Flooring, the user walks around their space placing points that define the bounds of the floor (See Figure 2a). Then the app creates a single, large polygon of flooring and draws it on top of the camera view. The user can walk around their physical space, viewing the flooring from different perspectives in augmented reality. However, since there is no occlusion handling, the illusion of augmented reality breaks when a real-world object should occlude the flooring but the flooring continues to be drawn in front of it (See Figure 2b). In terms of immersiveness, the AR Home Flooring app is more immersive than Roomvo since it allows users to walk around a space in real-time, but less immersive than Planner5D due to this occlusion issue. In terms of ease of use, this app is easier to use than Planner5D since it doesn't require a full CAD model, but harder to use than Roomvo since it requires the user to manually segment the flooring.

With FAR, I have created a system which improves on the state of the art by automatically segmented the flooring in real-time, providing an ease of use similar to that of Roomvo and improving the visualization's immersiveness by providing an augmented reality experience with object occlusion handling better than apps like Planner5D and AR Home Flooring.

Application Overview and Usage

The Flooring in Augmented Reality app strives to provide a user-friendly experience, requiring no manual setup. As soon as users open the app, FAR begins searching for the ground plane and will render alternate floor covering as soon as the ground plane has been found. As users walk around their space, the app will automatically detect new flooring areas and render the flooring in the space (See Figure 3c). Additionally, as users navigate their space and pass behind objects which occlude the floor, FAR ensures that the floor rendering will be occluded as well, maintaining the augmented reality illusion. Lighting conditions on the flooring also dynamically update as they change in real life, allowing users to understand how light and shadow affect the flooring at different times of day or with different lights dimmed.

By tapping on the grid icon in the corner, users can select from different flooring types including brick, marble, wood, and carpet (See Figure 4a). While the flooring catalog currently has only a couple products, it provides a good starting point for other developers to add their own. This menu also gives users the ability to rotate the flooring to align it better with their spaces.

Under the gear icon, users can find developer-related settings for doing things such as turning on processing statistics, enabling/disabling the lighting simulation, and switching segmentation models to tune the balance between quality and performance (See Figure 4b).

Design Decisions

The following goals guided the design decisions that went into the FAR system:

1. Easy setup and use. The app should automatically find and replace the flooring in a space without requiring the user to specify the flooring bounds.
2. High immersiveness. The app should run in real-time, allowing the user to walk around a space to view flooring from different angles.
3. High quality visualization. The app should accurately detect the boundaries between flooring and other surfaces and correctly replace flooring, adjusting for potential surface occlusions. Lighting and shadows should be approximated as well to improve realism.

In the following sections, I'll discuss the major components of the FAR system and how the choice to use those components fulfills the goals mentioned above. Before explaining each component of the FAR system in detail, however, it will be helpful to explain how each of the components fit into the overall system. As seen in Figure 5, there are many steps in the pipeline to produce the final augmented reality image.

The system works by leveraging Unity's AR Foundation to track the camera's position and recognize the ground plane. When the camera moves beyond a certain threshold, the camera's image is passed to a DeepLab semantic segmentation network hosted in an Android Java plugin to generate a rough segmentation of the floor asynchronously. This segmentation is further processed by a watershed segmentation algorithm to refine the segmentation. Once this is complete, the input image and segmentation are passed to a Lighting Extractor to create a texture which simulates the lighting conditions on the floor. Then the segmentation image and lighting simulation image are passed back to Unity's main thread and used as textures in Projectors to cast the segmentation mask and lighting texture onto the floor plane. This allows FAR to clip the floor plane to the segmentation mask and alter the floor plane's brightness according to the lighting texture, simulating a flooring replacement in augmented reality.

Augmented Reality

To ensure the system is as immersive as possible, I built it as an augmented reality app that could run on a mid-range Android phone. I chose to use the Unity game engine as the basis of the app since Unity provides an efficient rendering pipeline that works well in mobile apps. Also, compared to Unreal Engine's C++ development, Unity's C# scripting is a lot more manageable for a small, one-man team, allowing me to develop faster than I would be able to with UE4 or a bespoke rendering engine.

Unity also provides the AR Foundation platform. AR Foundation provides a layer of abstraction above Android's ARCore or Apple's ARKit augmented reality platforms, simplifying development for both types of devices. FAR leverages AR Foundation to track the phone's movement and render the flooring in perspective.

To locate the flooring plane in 3d space, FAR uses AR Foundation's Plane Detection and Trackable APIs. As the phone moves around in a space, AR Foundation automatically detects and tracks planar surfaces, expanding the planes as it gathers more data about the environment. While AR Foundation's surface tracker is usually accurate, it occasionally detects surfaces incorrectly and will generate planes far above or below the actual physical surface. In order to correct for the occasional errant plane, FAR searches through all of the recognized horizontal planes every couple of frames and select the plane with the largest surface area. During testing, this method correctly selected the flooring plane almost every time. In addition to filtering incorrect planes, this strategy allows the app to correct its estimate of the floor position in cases where the user starts the app with the camera pointing at a higher surface such as a table and then later points it at the ground. Since AR Foundation is constantly detecting and expanding its tracked planes, the FAR app's surface tracker is able to recognize a new, larger surfaces and adjust the floor plane's position accordingly.

Segmentation Projector

In FAR, the floor is simply represented as a large plane with a carpet texture repeated across it. The surface tracker mentioned in the previous section positions this plane at the height of the real-world floor, but it still appears as a large surface extending out to the horizon. To mask the plane to the boundary of the real-world floor, FAR uses Unity's Projector object along with custom clipping shaders. In Unity, the Projector is a special object which functions similar to a projector in a movie theater. It applies a specified material (a shader and its associated properties such as textures) to all objects which intersect its frustum so that the material appears projected onto the objects' surfaces.

During runtime, AR Foundation keeps track of the phone's physical location. If the phone translates or rotates by a large enough distance, FAR will take a snapshot of the phone's current transform as well as the current image that the phone's camera captured. Since creating the segmentation mask can take around 700ms (too much time to process in a single frame), FAR starts an asynchronous task to process the camera image. This processing will be explained in detail in the next section. Once the processing task is complete, the segmentation mask is assigned to the segmentation projector's material and the projector is set to the position and orientation of the previously captured snapshot. Using this method, the segmentation mask is projected onto the floor plane from the same position that the initial image was captured.

While segmentation processing cannot be performed in real-time, the fact that the segmentation mask is projected onto the floor with the same projection that the camera captured means that the floor will appear correctly masked even after small camera movements. The movement threshold at which FAR calculates a new segmentation is set to balance the need for quick updates for a smooth experience and the need for image stability so that segmentation borders don't jitter with every slight movement and subsequent mask calculation.

In order for the segmentation projector to clip the flooring plane using the calculated segmentation mask, FAR uses a custom shader. This shader takes as input the segmentation mask, the segmentation projector's projection matrix, and the vertices of the surfaces which intersected the projector's frustum. In the vertex shader function, the shader modifies the surface vertices' UV coordinates according to the projector's projection matrix. Then in the fragment shader function, the shader samples the segmentation mask and writes to the screen's depth buffer anywhere flooring shouldn't be present. Where flooring should exist, the shader discards the pixel/fragment, preventing writing to the depth buffer. By writing to the depth buffer in places where carpet shouldn't exist, the plane is hidden from view, allowing Unity's AR view of the physical world to be displayed on screen.

Since the camera can be moved faster than it is possible to process floor segmentation, it is possible for the user to see the floor plane outside the frustum of the segmentation projector. FAR's segmentation projector shader handles this case by repeating the edge pixels' values beyond the edges of the projector. This way, if the edge pixel contains the floor, FAR assumes the floor continues beyond the edge of the segmentation, and if the edge pixel does not contain the floor, FAR assumes there's not going to be more floor beyond that edge (See Figure 6). In practice, this method works very well at providing an accurate segmentation outside the bounds of the currently captured image, improving the perceived performance of the app.

Semantic Segmentation Model

A primary goal for FAR is that the augmented reality system provides an ease of use comparable to that of static image visualizers such as Roomvo. That means that the system must automatically identify the flooring surface without any user intervention. This automatic, real-time segmentation proved difficult on mobile devices with limited resources. While classical approaches such as watershed image segmentation are generally performant enough for mobile devices, they require the user to specify which areas of the image are initial guesses for the segmentation. In situations where the flooring occupies the lower half of the image and walls and ceiling occupy the upper half, it is possible to automatically specify initial segmentation guesses. This approach works well in tightly controlled scenarios, but the augmented reality use case provides the additional challenge that the image may not neatly fit into this pattern. Handling the more complex images that arise in augmented reality scenarios would require making more conservative initial guesses in watershed segmentation. By specifying smaller regions as initial guesses for segmentation, the algorithm would be more likely to incorrectly segment the floor from the rest of the image, leading to poor results. Other approaches to an initial segmentation guess which are not based on location in an image but properties of the image such as color or texture also lead to poor results. Since flooring can be any color or pattern and take up any percent of the image, there is not a reliable heuristic to use when estimating the initial segmentation labels.

Fortunately, semantic segmentation neural networks are able to “understand” the content in an image much better than classical segmentation algorithms. Unfortunately, however, these networks are generally too resource intensive to run in real-time on mobile devices. A large portion of time was spent researching efficient segmentation networks which could run with limited resources. Initial attempts with U-Nets (Ronneberger, Fischer, & Brox, 2015) and E-Nets (Paszke, Chaurasia, Kim, & Culurciello, 2016) worked well in a desktop context, but the networks took around 20 seconds to perform one inference on a mobile device. Since FAR is intended to be a real-time visualization solution, that performance was unacceptable.

The most performant network I found (and the network I’m currently using) is a TensorFlow implementation of DeepLabV3 (Chen, Zhu, Papandreou, Schroff, & Adam, 2018) with MobileNetV2 (Sandler, Howard, Zhu, Zhmoginov, & Chen, 2018) as its feature extractor. While the previous networks were large and powerful, DeepLabV3-MobileNetV2 was optimized by Google for on-device inference which leads to significant performance boosts. When performing semantic segmentation on a mobile device, DeepLab performed an inference in

700ms, a significant improvement over the approximately 20 seconds it took for other networks. While this speed isn't fast enough to perform segmentation every frame (a requirement of ~30ms/inference), it is fast enough to use with the segmentation projector system described previously.

I made attempts to increase the performance of the DeepLab network by decreasing the network's output size from 513x513 pixels to 257x257 and 129x129. With the reduction in size came considerable performance gains, reaching speeds of 240ms/inference for the 257x257 model and 110ms/inference for the 129x129 model. However, the quality of segmentation was reduced in the process as well. Results from the 257x257 model were decent, but results from the 129x129 model were unusable. I opted to use the 513x513 model because it provided the best segmentation results while still being fast enough for my use case.

One of the most labor-intensive parts of working with neural networks is finding and collecting a dataset for the specific task. Fortunately, the OpenSurfaces dataset (Bell, Upchurch, Snavely, & Bala, 2013) and the ADE20K dataset (Zhou, et al., 2018) were available for use. The OpenSurfaces and ADE20K datasets provide vast collections of images that are segmented and annotated with surface properties such as material and texture. Since the datasets include large amounts of information unrelated to the FAR project and their data is stored in different formats, the data needed to be preprocessed before using it to train the DeepLab network. A script filtered the images, keeping only those that had more than 10% of the total image labeled with carpet, tile, or linoleum. Images that met this criterion were representative of the conditions that would exist in the real-world scenarios the FAR app would face. After filtering the images, the script converted the carpet, tile, and linoleum labels to a single "flooring" value and every other label to a separate "background" value. Next, it scaled and padded the images to be uniformly

512x512 pixels in size. Then, it performed data augmentation by also making a horizontally mirrored copy of each image. I considered vertically mirroring or rotating the images as well, but decided against it since I didn't want the network accidentally segmenting the ceiling or walls as if it were the floor. Finally, the script split the resulting images into training and test sets and converted them into the TFRecord format, an efficient data format for training TensorFlow networks. Using these filtering and data augmentation techniques, the final combined dataset contained 18982 training images and 386 test images.

Since it takes such a long time to train the DeepLabV3-MobileNetV2 network from scratch, transfer learning was used to pretrain the model with weights learned from the PASCAL VOC training set (Everingham, et al., 2015). Then, the network weights were fine-tuned to correctly segment flooring based on the combined dataset. Since the DeepLab network was initialized with pre-trained weights, it converged quickly. The network only needed to train for around 2000 steps with a batch size of 16 before training loss stopped decreasing.

After training, I ran the network on my test images in order to visually confirm that the network produced accurate segmentations of flooring. A script froze and exported the network graph, performing post-training dynamic range quantization, a technique which converts 32-bit floating point weights to 8-bit integers in order to save space and improve efficiency.

Then, the script exported the network as a TFLite file. Although the DeepLab network was trained using a Python framework, it's not possible to run the Python code on an Android device. TensorFlow Lite bridges that gap by allowing the trained network graph to be exported as a TFLite file which can be run by the TensorFlow Lite interpreter on mobile devices.

Since the TensorFlow Lite interpreter plugin for Unity is still in an experimental, unstable state, I decided to forgo using it and opted to write an Android AAR library to host the interpreter myself. The Android library could then be imported into Unity and used like an ordinary Android plugin. While there is a cost to marshalling data between Unity and Android plugins, the cost was mitigated by only sending camera data to the Android library when it was time for a new segmentation and to only send the final results back to Unity.

Segmentation Refinement

Since the DeepLab network outputs a very rough segmentation mask, an additional refinement step is needed to produce an accurate segmentation. The refinement step works by taking the segmentation model's approximate mask and performing morphological erosion and dilation to create 3 labeled regions: "known flooring", "known background", and "unknown". The "known flooring" is the region identified by the segmentation model inset by several pixels, while the "known background" is the region unlabeled by the segmentation model inset by several pixels. The "unknown" region is the band of pixels between the flooring and background regions (See Figure 7). The refiner uses this mask of known and unknown regions, along with the camera image used in semantic segmentation, as input to the watershed segmentation algorithm. The algorithm expands the known regions outward, placing the boundary between regions along large discontinuities in the image which usually end up being the boundary of the floor. Because there is only a very small region of unknown pixels for the watershed algorithm to segment, it performs the task with very low latency and high accuracy.

Lighting Extraction

To sell the illusion that the floor exists in the physical world and is not simply a plane floating in space, the flooring surface must be receptive to the lighting conditions in the real world. To simulate these lighting conditions, FAR uses a Unity Projector similar to the technique used for clipping the floor plane. Each time the floor segmentation mask is updated and the segmentation projector is moved, a lighting texture is generated and the lighting projector is moved to the same position as the segmentation projector. A shader similar to segmentation projector's shader projects this lighting texture onto the flooring surface. Instead of masking the floor with a depth buffer however, this shader adjusts the brightness of each pixel by darkening pixels where the lighting texture is darker than 50% gray and brightening pixels where it is lighter than 50% gray.

The lighting texture itself is generated by passing the camera image along with the segmentation mask into a Lighting Extractor after the image segmentation process has completed. The Lighting Extractor works by converting the camera image from RGB to grayscale. Then, in order to preserve overall lighting effects and ignore smaller details in the existing carpet patterns, the image is blurred. Since different carpet patterns can naturally be lighter or darker, the average brightness of the masked flooring is taken and used to offset the whole lighting image so that the average brightness is 50% (See Figure 9 for lighting extraction steps).

In situations where there is a large contrast difference along the edge between carpet and walls, the brightness of the walls can bleed into the edge of the carpet during blurring which can cause a halo effect (See Figure 8 for an example of the halo issue). FAR combats this issue with additional processing steps in the Lighting Extractor. Before blurring the camera image, the segmentation mask is used to set all non-floor pixels to black. A form of masked blurring is implemented by blurring the segmentation mask the same amount as the camera image and then dividing the blurred camera image by the blurred segmentation mask. This has the effect of weighting the blurred camera image to ignore non-floor pixels, countering the halo effect.

By generating this lighting texture and using a lighting projector to alter the brightness of floor pixels, a convincing lighting simulation is achieved which drastically improves the realism of the augmented reality app (See Figure 3 and Figure 10 for several examples of the impact of lighting simulation).

Validation and Future Work

As you can see in Figure 11*a* and *b*, the FAR system does a good job at segmenting the flooring in spaces which have relatively plain walls and floors. In these spaces, the segmentation border cleanly follows the edges of the ground since there is a clear distinction between the two surfaces. However, in spaces which have large amounts of clutter (see Figure 11*c*), the semantic segmentation model occasionally struggles to identify flooring. This isn't a big problem though, because the system is intended for use in customer-facing commercial spaces which most likely won't have as many distractions. A more frequent issue shown in Figure 11d is the failure of FAR to correctly segment the ends of long hallways. The issue lies in the fact that the DeepLab-MobileNet network cannot generate long, thin segmentations, so the end of the long hallway is not included in the rough segmentation mask. Potential solutions to this problem include finding alternative segmentation networks or improving the segmentation refinement step. Currently, FAR uses watershed segmentation to refine the mask due to its speed. However, more sophisticated segmentation methods could be explored in order to find a technique which better classifies foreground and background based on the supplied rough segmentation mask.

An additional area for future work could be focused on flooring products and layouts. Currently, there is a limited selection of flooring products available. This is due to the fact that this project's focus is to create a proof of concept for an augmented reality flooring solution and not a finished, shippable product. Further work is needed to integrate specific product lines for more rendering options. Additionally, the products that are included can only be displayed using a standard grid layout. More complex layouts including half brick, half drop, and quarter turn have yet to be implemented. Since this wouldn't be a difficult task to complete, I focused my development efforts on the more complex tasks of segmentation and lighting approximation.

Conclusion

Although FAR is still in the prototype phase of development, it succeeds at improving on the state of the art in floor covering visualization. The app sets the bar for ease of use of augmented reality flooring viewers by automatically locating and segmenting the floor surface without requiring any user intervention through the use of real-time semantic segmentation networks. The app is also often on par with the visual fidelity of static image systems, properly masking the floor and estimating lighting conditions in most cases, leading to high quality, realistic results. Overall, the app serves as a strong, innovative foundation for future development in augmented reality for flooring.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., . . . Jia, Y. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Retrieved from https://www.tensorflow.org

AR Home Flooring. (2019, Jan 9). *1.0.2*. Ensena Soft S.A. de C.V. Retrieved Aug 6, 2021, from https://play.google.com/store/apps/details?id=com.ensenasoft.arhomeflooring

Bell, S., Upchurch, P., Snavely, N., & Bala, K. (2013). OpenSurfaces: A Richly Annotated Catalog of Surface Appearance. *ACM Trans. on Graphics (SIGGRAPH), 32*. Retrieved Sep 9, 2021, from http://opensurfaces.cs.cornell.edu/

Beucher, S., & Meyer, F. (1993). Segmentation: The Watershed Transformation. Mathematical Morphology in Image Processing. *Optical Engineering, 34*, 433-481. Retrieved Oct 27, 2021

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*. Retrieved Sep 22, 2021, from https://github.com/opencv/opencv

Chen, L.-C., Zhu, Y., Papandreou, G., Schroff, F., & Adam, H. (2018). Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation. *ECCV.* Retrieved Oct 19, 2021, from https://github.com/tensorflow/models/tree/master/research/deeplab

Everingham, M., Eslami, S. M., Van Gool, L., Williams, C. K., Winn, J., & Zisserman, A. (2015). The Pascal Visual Object Classes Challenge: A Retrospective. *International Journal of Computer Vision, 111*, 98-136.

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., . . . Adam, H. (2019). Searching for MobileNetV3. *ICCV.* Retrieved Oct 19, 2021, from https://github.com/tensorflow/models/tree/master/research/deeplab

Leap Tools Inc. (2021). *Roomvo*. Retrieved Aug 6, 2021, from https://www.roomvo.com/

Paszke, A., Chaurasia, A., Kim, S., & Culurciello, E. (2016). ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation.

Planner 5D. Interior Design: Room, Home, Floorplan. (2015, May 27). *1.26.26*. Planner 5D. Retrieved Oct 28, 2021, from https://play.google.com/store/apps/details?id=com.planner5d.planner5d

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. (N. Navab, J. Hornegger, W. M. Wells, & A. F. Frangi, Eds.) *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2015*, 234-241.

Rother, C., Kolmogorov, V., & Blake, A. (2004). GrabCut - Interactive Foreground Extraction using Iterated Graph Cuts. *ACM Transactions on Graphics (SIGGRAPH)*.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. *CVPR.* Retrieved Oct 19, 2021, from https://github.com/tensorflow/models/tree/master/research/deeplab

tilo Floor Layer. (2020, Apr 7). *1.7*. tilo GmbH. Retrieved Aug 6, 2021, from https://play.google.com/store/apps/details?id=com.tilo.app

Zhou, B., Zhao, H., Puig, X., Xiao, T., Fidler, S., Barriuso, A., & Torralba, A. (2018). Semantic Understanding of Scenes through the ADE20K Dataset. *International Journal on Computer Vision (IJCV)*.

Figures

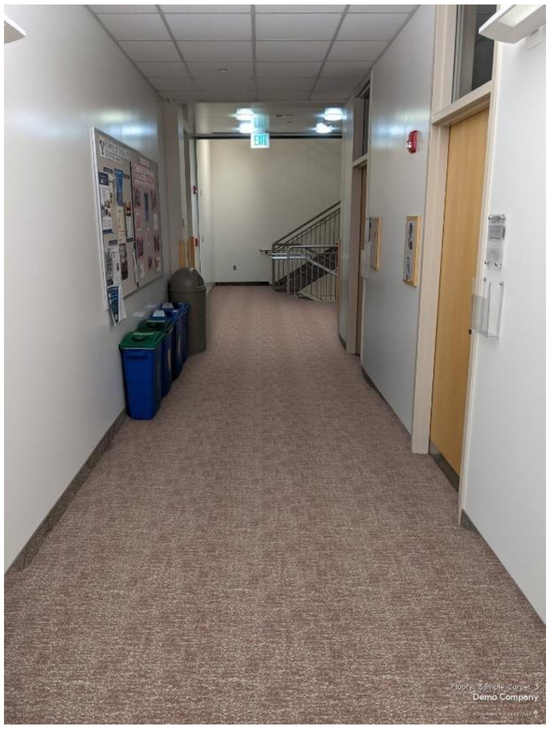
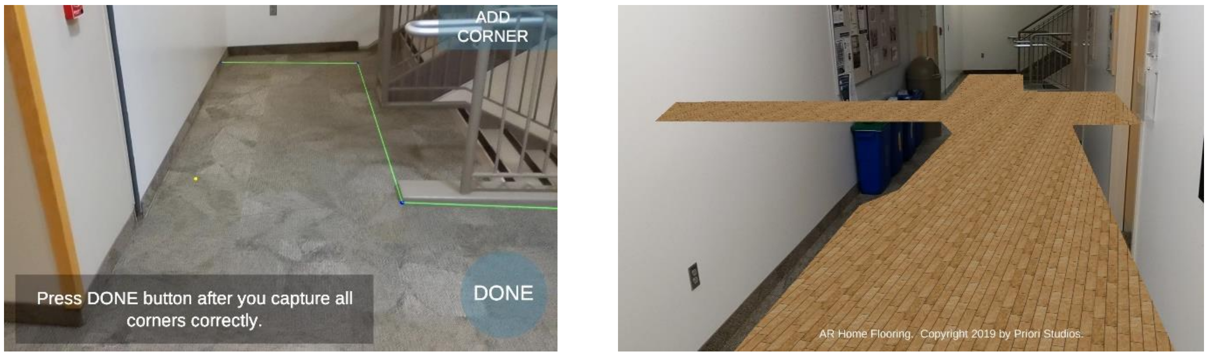


Figure 1. An example of the output created by Roomvo's visualizer. Notice the clean flooring segmentation and lighting approximation that were achieved with no user input. The quality is impressive, but the system only works on static images.


ADD CORNER
Press DONE button after you capture all corners correctly.
DONE
AR Home Flooring. Copyright 2019 by Priori Studios.

(a) *(b)*

Figure 2. AR Home Flooring example. (a) Demonstration of the process of manually placing room extents corner by corner. (b) The final result in augmented reality. Notice that the lack of lighting simulation and the lack of flooring occlusion break the augmented reality illusion.

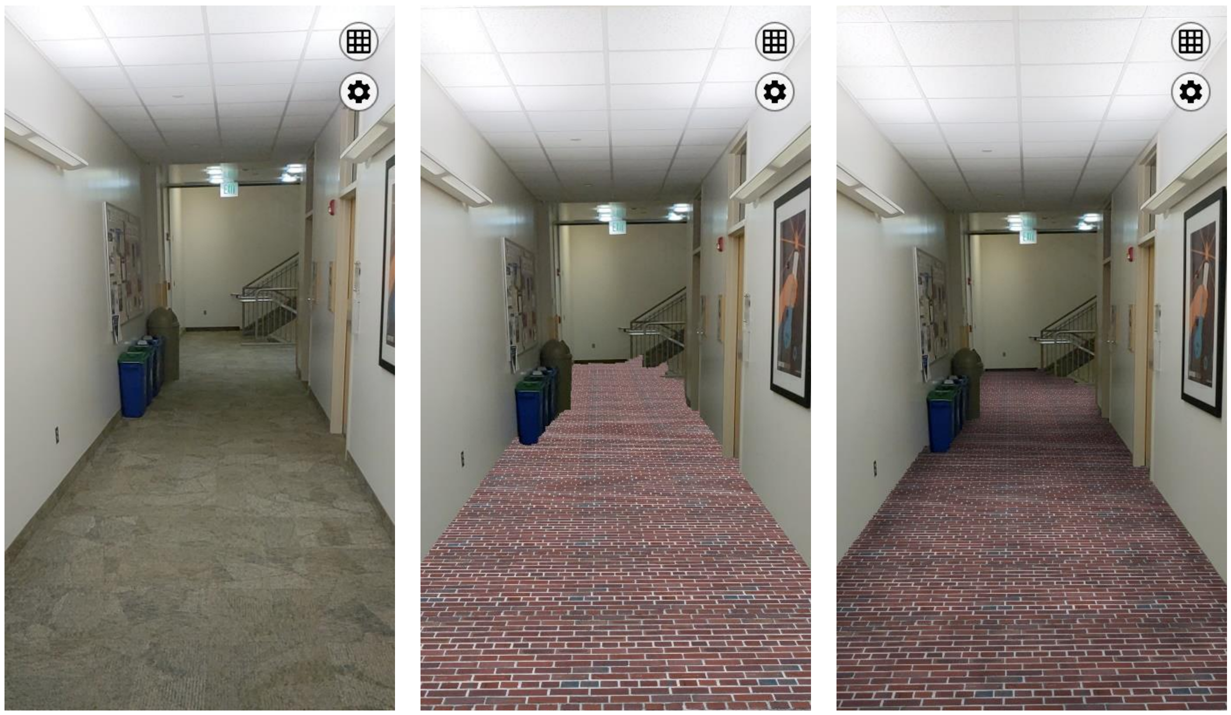
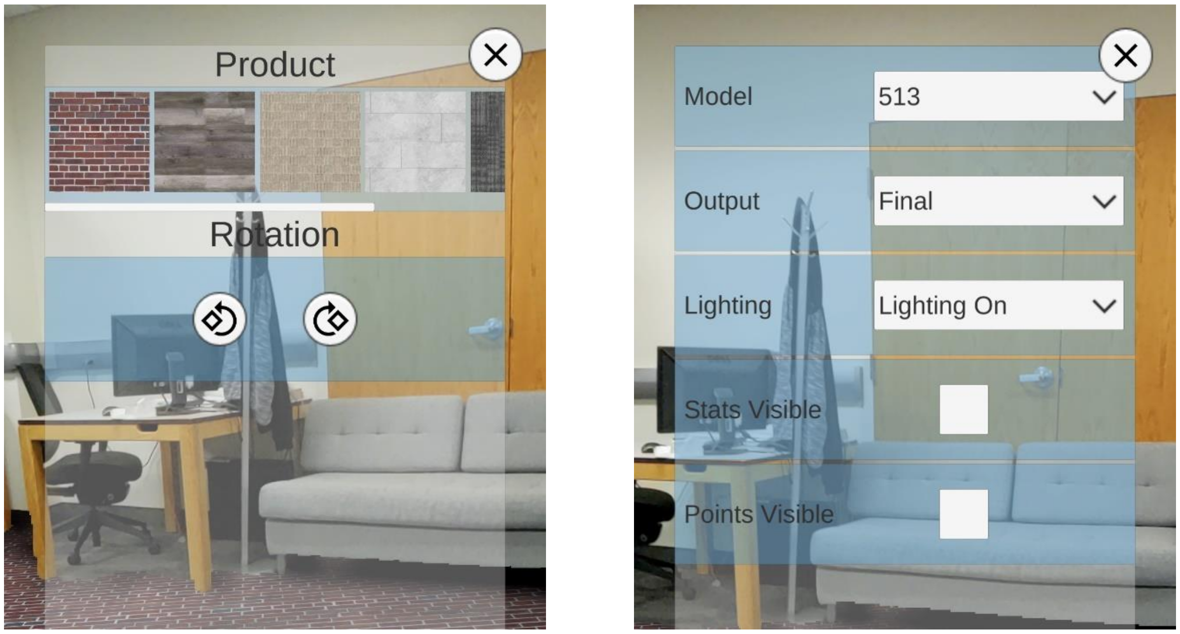


Figure 3. An example of FAR's visualization. (a) Original hallway. (b) FAR's output with lighting disabled. (c) FAR's output with lighting enabled.



Model 513
Output Final
Lighting Lighting On
Stats Visible
Points Visible

Figure 4. Menus in FAR. (a) Switching and rotating products. (b) Developer-oriented settings.

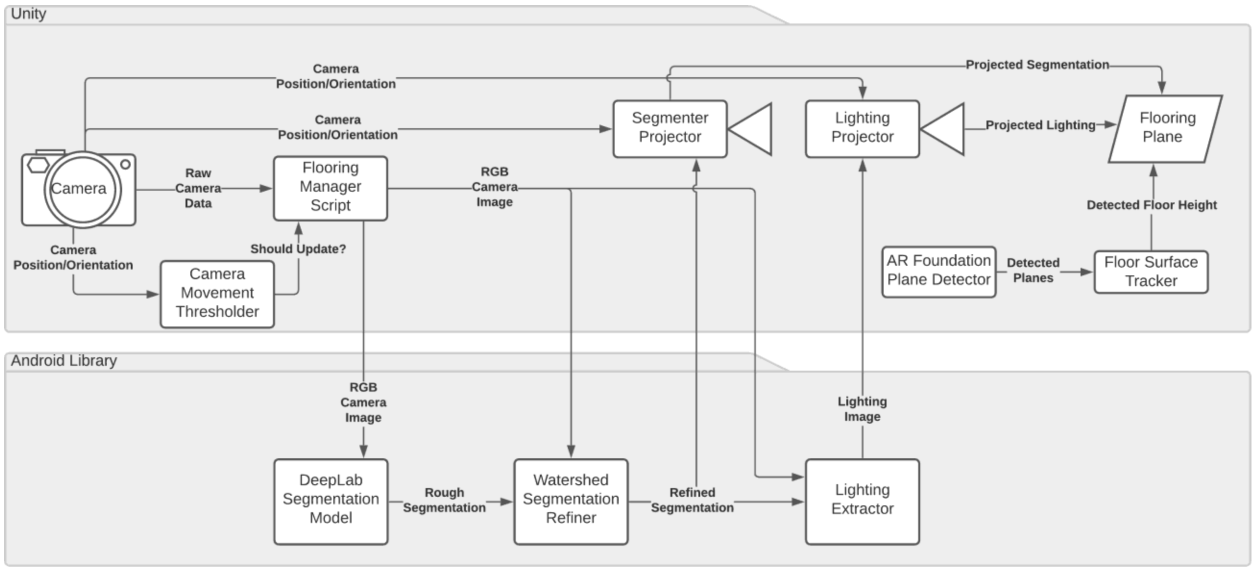


Figure 5. A high-level view of the data flow in FAR.

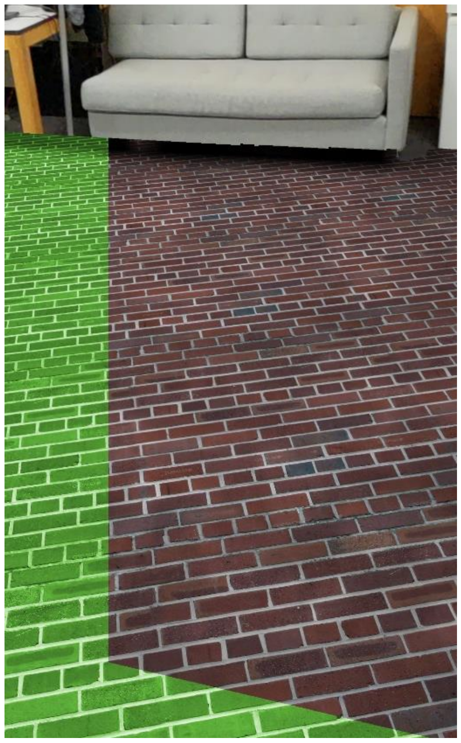


Figure 6. Visualization of the segmentation projector repeating the mask's edge pixels outside the projector's frustum bounds. The normal-colored floor represents the part of the image that was captured for segmentation. The green floor represents the pixels which were repeated out from the edges of the segmentation mask. Although the border isn't perfect, it is a very good estimation of which pixels contain the floor.

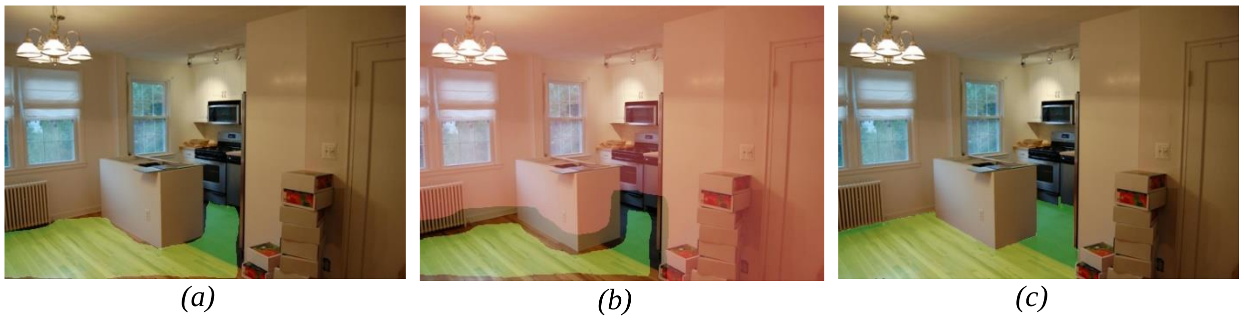


Figure 7. Watershed segmentation refinement. (a) The original segmentation prediction by the neural net. (b) The initial watershed segmentation labels. Green represents the "known flooring" region, red represents the "known background" region, uncolored represents the "unknown" region. (c) The refined segmentation output by watershed.

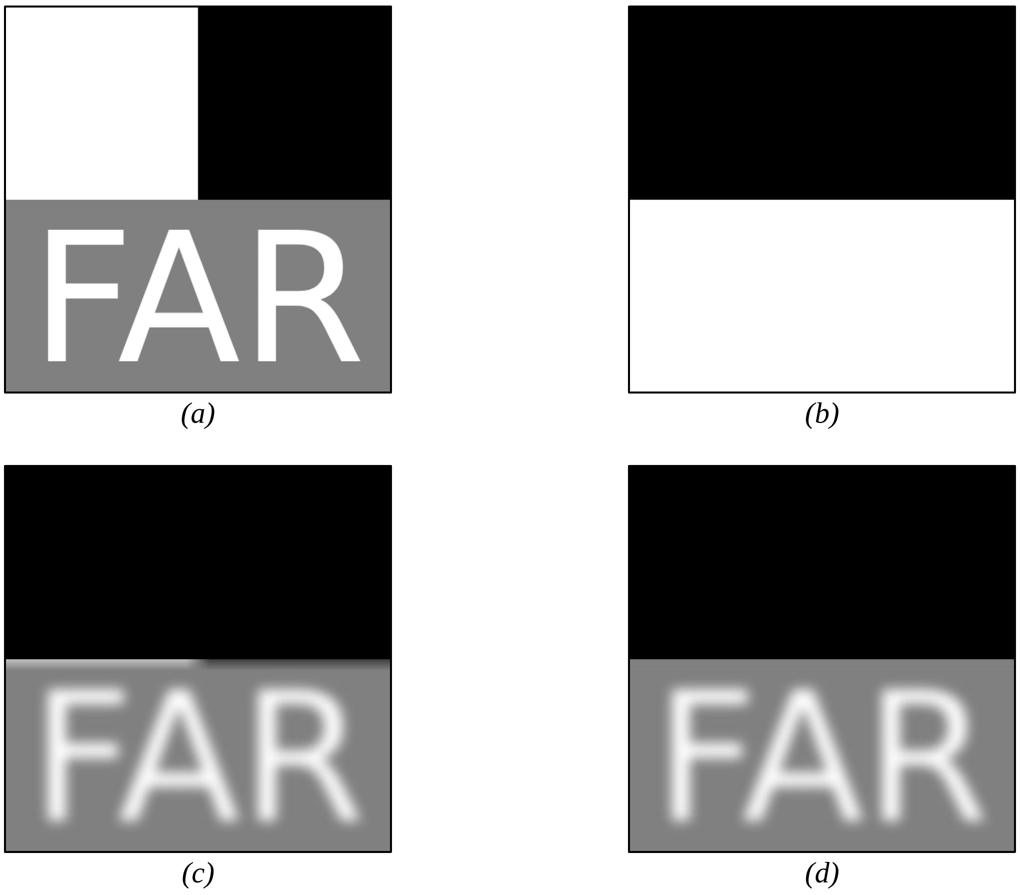


Figure 8. Example of the halo issue on borders when blurring. (a) Input image. (b) Input mask. (c) Incorrect halo artifact along output border due to naïve blurring and masking. (d) Correct result when inverse weighting by the blurred mask.

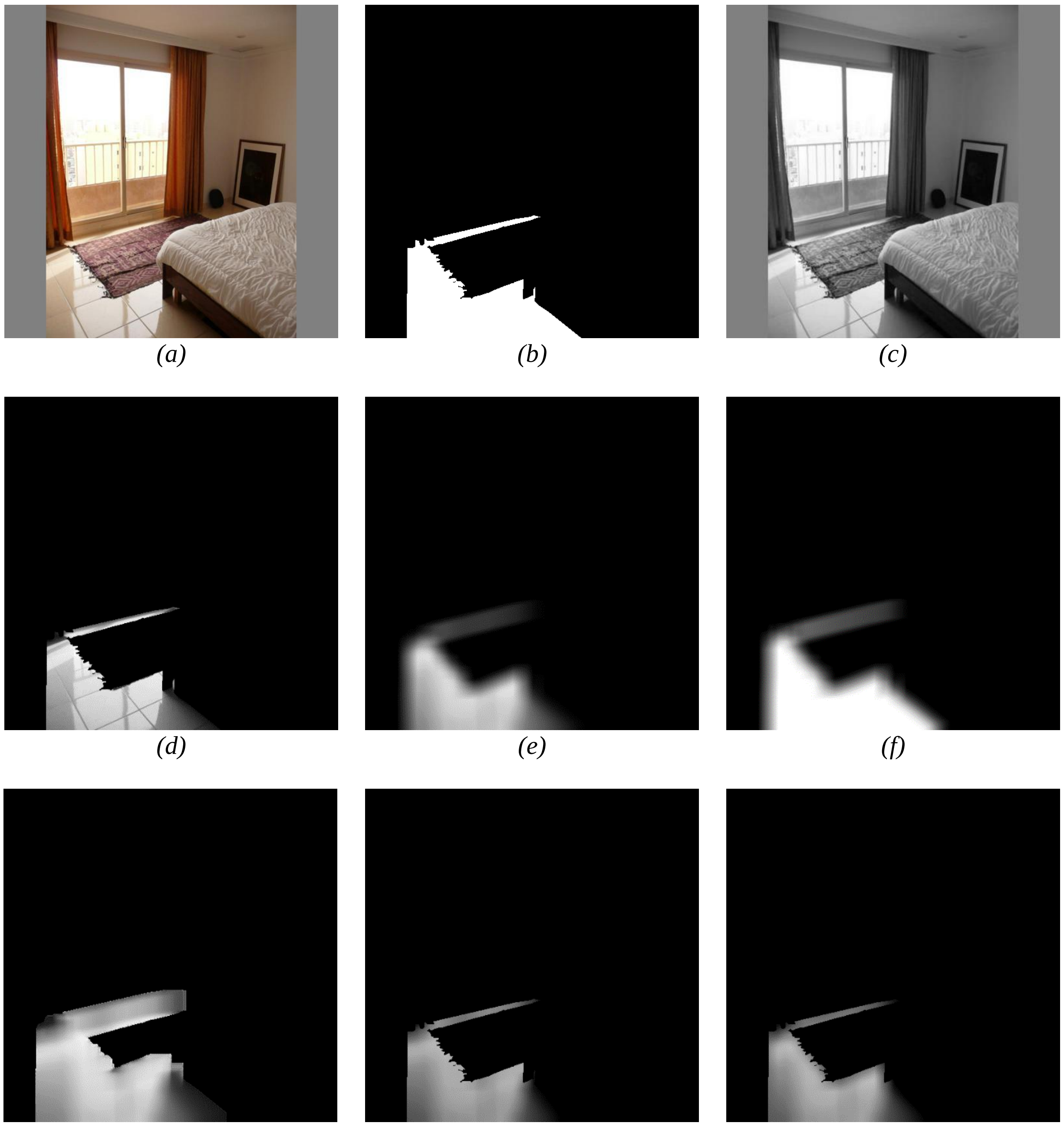


Figure 9. Steps for lighting extraction. (a) Original image. (b) Segmentation mask. (c) Image converted to grayscale. (d) Grayscale image masked by segmentation mask. (e) Masked image blurred. (f) Segmentation mask blurred. (g) Masked, blurred image inversely weighted by blurred segmentation mask. (h) Brightness normalized result. (i) Lighting result without accounting for halo issue. Notice the dark edges around the rug due to the rug's brightness bleeding over.

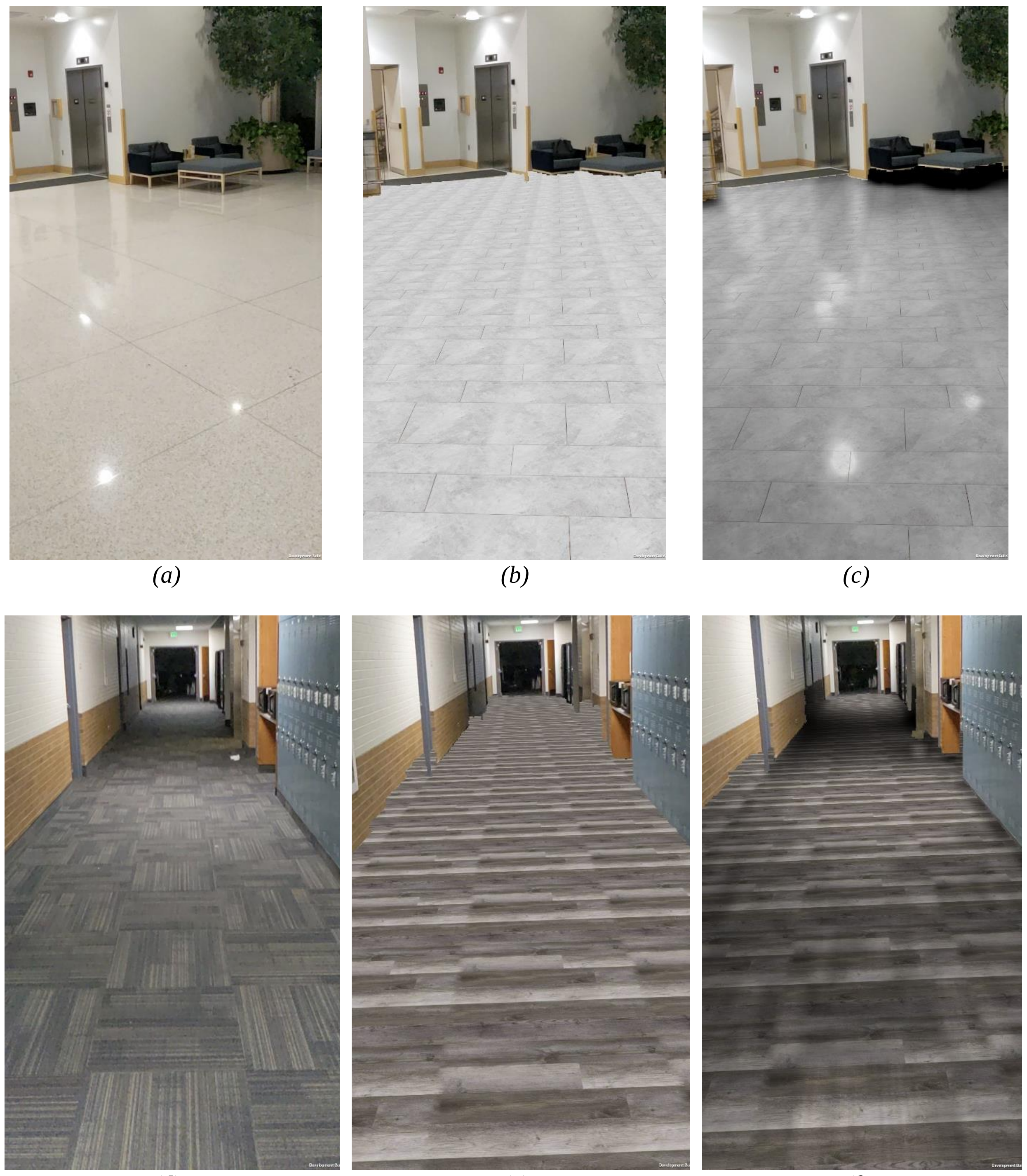


Figure 10. Examples of FAR results with and without lighting. (a and d) Original photo. (b and e) Rendering without lighting. (c and f) Rendering with lighting.

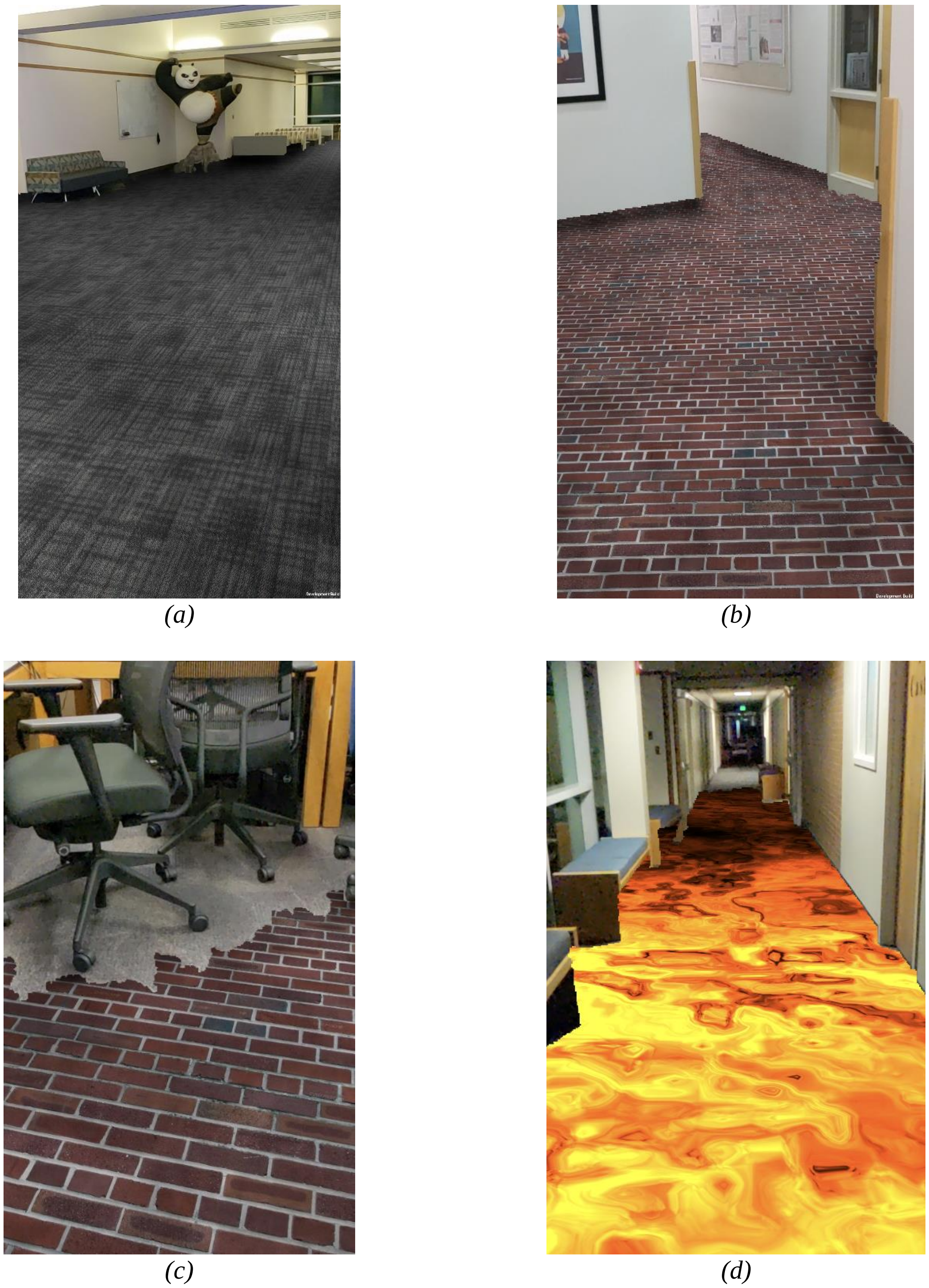


Figure 11. *Successful and unsuccessful results. (a and b) Successful segmentations with simple segmentation borders. (c) An unsuccessful segmentation due to a very complicated segmentation border. (d) A partially unsuccessful segmentation due to the hallway extending far into the distance.*

Time Log

			Total Complete:	
			288:22:00	
Date	Time In	Time Out	Duration	Notes
8/7/2021	12:45:00 PM	3:30:00 PM	2:45:00	Install Unity, AR Foundation Tutorial
8/7/2021	6:30:00 PM	9:30:00 PM	3:00:00	Continuing tutorial, bashing head against wall to debug unity on android
9/3/2021	3:00:00 PM	7:45:00 PM	4:45:00	Initial tracking of cube and plane
9/7/2021	4:00:00 PM	7:15:00 PM	3:15:00	Depth occlusion testing
9/9/2021	5:15:00 PM	8:30:00 PM	3:15:00	OpenSurfaces, research machine learning, python environment
9/9/2021	10:30:00 PM	11:30:00 PM	1:00:00	
9/10/2021			0:45:00	Time spent throughout the day doing a little bit at a time. Processing data
9/11/2021	12:30:00 PM	4:45:00 PM	4:15:00	UNets
9/13/2021	2:15:00 PM	8:30:00 PM	6:15:00	UNets
9/15/2021	11:15:00 AM	6:00:00 PM	6:45:00	UNets on campus. Tried with single data point and after many epochs finally converged to decent result but not amazing
9/16/2021	7:00:00 PM	8:15:00 PM	1:15:00	unets with fuzzed data and minibatches
9/17/2021	7:00:00 PM	8:30:00 PM	1:30:00	unet and trying to get remote desktop to work. Network never trained to a decent result.
9/17/2021	9:29:00 PM	11:59:00 PM	2:30:00	AR averaged height, depth occlusion. Depth occlusion is far too noisy to be useful. Maybe I really need to use deep learning
9/20/2021	11:30:00 AM	9:00:00 PM	9:30:00	Trying to get ubuntu 20.04 installed on seans computer... Training ENet
9/21/2021	5:44:00 PM	11:59:00 PM	6:15:00	Enet, and classical approach of watershed, unity3d on linux
9/22/2021	11:15:00 AM	4:30:00 PM	5:15:00	Enet is working! I was using jpgs and not pngs...
9/22/2021	4:44:00 PM	11:59:00 PM	7:15:00	Getting OpenCV to work on unity android, training net
9/27/2021	11:00:00 AM	2:15:00 PM	3:15:00	Graphs for validation/training, export traced model, java plugins for Unity3d
9/29/2021	11:00:00 AM	12:00:00 PM	1:00:00	Analyzing Validation loss
10/4/2021	10:45:00 AM	11:45:00 AM	1:00:00	
10/8/2021	9:00:00 AM	9:30:00 AM	0:30:00	Meeting
10/13/2021	2:45:00 PM	7:00:00 PM	4:15:00	Getting net to run in android plugin for unity
10/15/2021	9:00:00 AM	9:30:00 AM	0:30:00	meeting
10/16/2021	2:00:00 PM	5:00:00 PM	3:00:00	threading with networks
10/16/2021	7:44:00 PM	11:59:00 PM	4:15:00	Tensorflow network. It is so much faster wow.
10/19/2021	11:10:00 AM	4:50:00 PM	5:40:00	Getting camera data to plugin

10/19/2021	5:44:00 PM	11:59:00 PM	6:15:00	Deeplab networks
10/20/2021	12:00:00 AM	12:45:00 AM	0:45:00	""
10/20/2021	11:30:00 AM	8:45:00 PM	9:15:00	""
10/20/2021	9:14:00 PM	11:59:00 PM	2:45:00	""
10/21/2021	12:00:00 AM	1:15:00 AM	1:15:00	""
10/21/2021	1:00:00 PM	11:59:00 PM	10:59:00	""
10/22/2021	12:00:00 AM	1:11:00 AM	1:11:00	""
10/22/2021	1:50:00 PM	4:45:00 PM	2:55:00	""
10/25/2021	12:25:00 PM	3:45:00 PM	3:20:00	Deeplab
10/26/2021	11:25:00 AM	2:25:00 PM	3:00:00	Flooring tiles
10/26/2021	6:59:00 PM	11:59:00 PM	5:00:00	Deeplab, opencv
10/27/2021	12:00:00 AM	5:15:00 AM	5:15:00	Watershed segmentation, grabcut segmentation
10/27/2021	1:00:00 PM	7:25:00 PM	6:25:00	"", clipping flooring in unity
10/27/2021	7:59:00 PM	11:59:00 PM	4:00:00	Refactoring and improving
10/28/2021	12:00:00 AM	2:30:00 AM	2:30:00	""
10/28/2021	9:30:00 PM	11:00:00 PM	1:30:00	Paper rough draft
10/29/2021	4:29:00 PM	11:59:00 PM	7:30:00	Paper rough draft
10/30/2021	12:00:00 AM	1:00:00 AM	1:00:00	Paper rough draft
10/30/2021	12:30:00 PM	5:00:00 PM	4:30:00	Paper rough draft
11/4/2021	4:20:00 PM	10:15:00 PM	5:55:00	
11/5/2021	8:10:00 PM	11:59:00 PM	3:49:00	Object tracking
11/8/2021	11:00:00 AM	6:00:00 PM	7:00:00	
11/9/2021	1:15:00 PM	7:00:00 PM	5:45:00	
11/11/2021	3:30:00 PM	7:50:00 PM	4:20:00	lighting, asyc segmentation
11/12/2021	8:14:00 PM	11:59:00 PM	3:45:00	settings
11/13/2021	12:00:00 AM	2:00:00 AM	2:00:00	settings, projection outside bounds
11/15/2021	10:59:00 AM	11:59:00 PM	13:00:00	lighting, improving repo
11/20/2021	12:45:00 PM	6:30:00 PM	5:45:00	
11/22/2021	11:00:00 AM	12:00:00 PM	1:00:00	
11/22/2021	12:30:00 PM	8:15:00 PM	7:45:00	lighting and cleanup
11/24/2021	12:30:00 PM	7:30:00 PM	7:00:00	lighting, ADE20K
11/29/2021	11:15:00 AM	7:30:00 PM	8:15:00	Fix lighting, fix surface tracker
11/30/2021	7:00:00 PM	11:59:00 PM	4:59:00	Flooring Options, Lava, ADE20K training

12/1/2021	12:00:00 AM	1:15:00 AM	1:15:00	
12/2/2021	7:35:00 PM	10:30:00 PM	2:55:00	
12/3/2021	3:45:00 PM	7:40:00 PM	3:55:00	Rotate Flooring, Anisotropic Filtering, Finding Memory leak/thrashing
12/4/2021	6:00:00 PM	11:30:00 PM	5:30:00	fixing leak slow down
12/6/2021	5:45:00 PM	11:15:00 PM	5:30:00	Attempt Tiles and fixing leak. nothing worked...
12/7/2021	2:00:00 PM	11:59:00 PM	9:59:00	Paper
12/8/2021	12:00:00 AM	3:30:00 AM	3:30:00	Paper figures
12/8/2021	11:30:00 AM	2:45:00 PM	3:15:00	Paper corrections