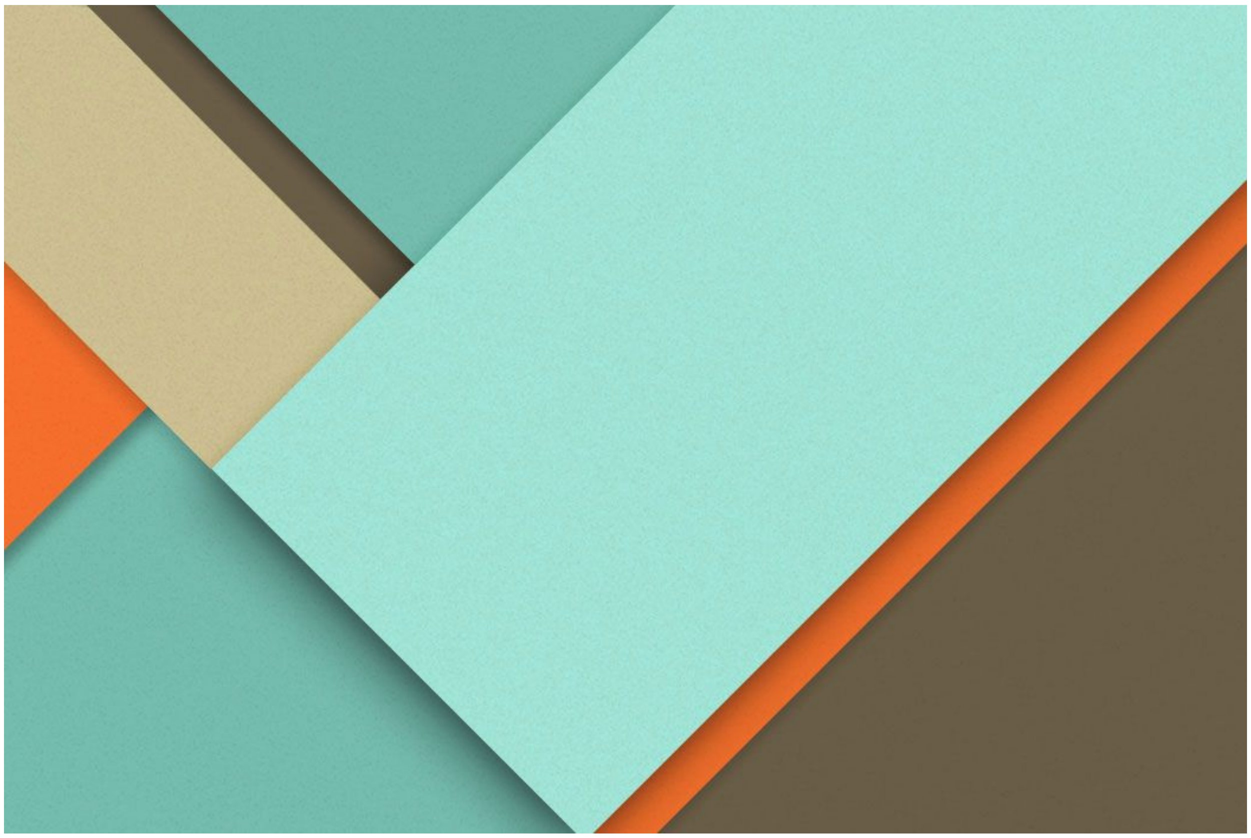


Undergraduate Library Web VR Experience

12.17.2018

Luke Smith, John Diffor, Sushrut Vani, Shaleen Agrawal, Jacob Carr
University of Illinois Urbana-Champaign
CS 498VR

Table of Contents

Overview

In this project, we built a walk-through model of the main floor of the Undergraduate Library as a Web VR experience through the use of Mozilla's state-of-the-art browser-accessible VR framework, A-Frame. The purpose of this model is two fold: namely, to attract alumni who may reminisce about the earlier appearance of the ever changing library, and secondly, for archival purposes. This model was built using high tech cameras and tools available through Media Commons.

1) Camera Setup

Step by step GoPro Fusion Instructions

1) Acquire the GoPro Fusion 360 Camera. A picture is shown below.

2) Make sure the camera is fully charged prior to use, and that two micro SD cards are inserted into the camera to be able to store the media contents.
3) Turn on the power, and ensure from the settings menu that the camera's WIFI connections are turned on.
4) Download the gopro app on your phone to remotely connect and take pictures from the camera. The Android link is here and the IOS link is here.
5) Slide from the left on the app to get to the main screen menu.
6) Hit the camera button and click the "+" symbol on the top-right side to add a camera. Note, the pairing process will not function correctly if you select the Fusion camera to pair with, we had to select the Hero5 camera. We were not able to determine exactly why the application only works this way, however, the process was repeatable across multiple smartphones.
7) Follow the onscreen instructions to successfully pair the camera.
8) Once your camera is paired, you may hit the shutter button on the phone and evaluate the result. A live preview should appear as well.

Additional Details

In order to capture the 360 images, we used a GoPro Fusion 360 camera. We also tried using a Nikon KeyMission 360, however we had many issues with this camera. As partially evidenced by the KeyMission 360's 2.1 out of 5 average review score on Amazon[1], there are a lot of problems with this camera. The primary issue we ran into was simply not being able to connect to the camera using the phone application even after multiple attempts with different phones. The GoPro Fusion 360 was slightly more user friendly, although we discovered there is a trick to getting it paired with the GoPro phone application. As explained above, selecting the Fusion camera to pair with does not work for an unknown reason, the user must select a different camera such as the Hero5, and then the pairing process should continue.

2) Image Quality

Lighting and Environmental Conditions

For the best viewing experience, it is better to have as little distractions and obstructions as possible. For this reason, we recommend shooting at a time when there is little to no foot traffic. While it is possible to shoot in the mornings, we do not recommend it. Sun rays change direction over the course of time, and as a result, shadows may be created which could lead to discontinuity and an unpleasant viewing experience. It is for this reason we recommend that all shooting be done at night, when there is no foot traffic present, and the lighting conditions remain continuous for a few hours. Additionally, the lighting inside the library is quite good, even during late hours.

Camera and Tripod Setup

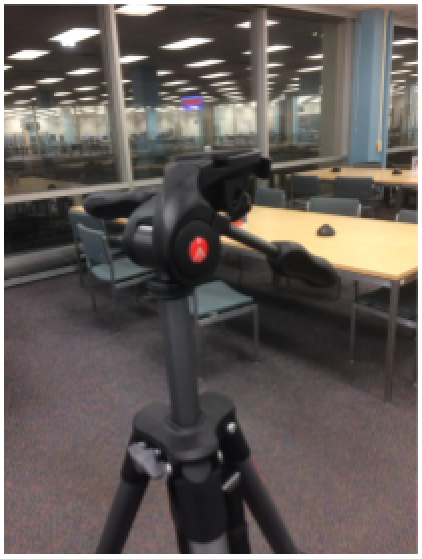


Figure 2.0

In order to accurately simulate the experience of someone walking through the library, a tripod was the clear choice to hold the camera. There are two main considerations one must keep in mind when selecting an appropriate tripod. First, since the camera is designed to take 360 degree photos, the tripod must have an extremely thin profile. The first tripod we tried, shown in figure 2.0, was too large and artifacts can be seen in the bottom of the images (see figure 2.1). We switched to a very thin tripod, it can be seen in figure 2.2 that almost no parts of the tripod are visible from above, this is a good indicator that clear pictures will be possible. Finally, since these images are intended to act as a virtual tour, we decided to set the height of the tripod to an average human height as shown in figure 2.3. We obtained the figure of approximately 5' 4" from a simple google search of average human height.

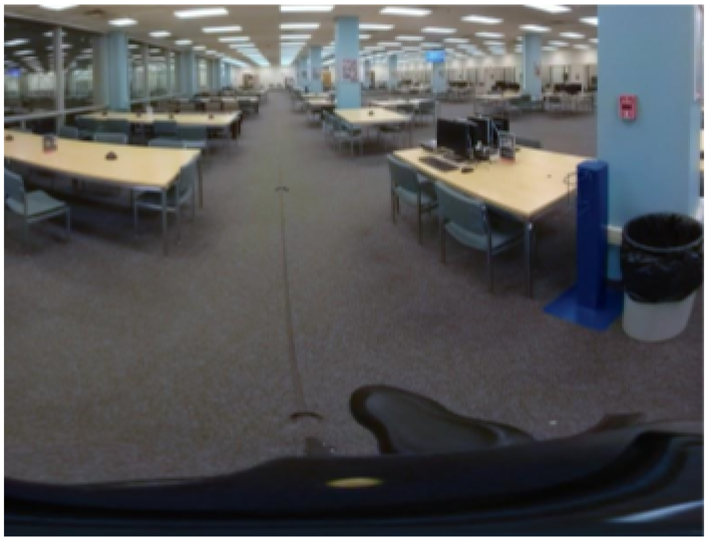


Figure 2.1

Figure 2.2

Continuity

An important aspect to keep in mind while shooting images for a virtual tour is to keep the distance between images consistent and in line with each other. This allows each transition to feel consistent for the user. As shown in figure 2.4, we used a tape measure and black tape to mark 10 foot intervals in between each picture. Although a shorter distance would result in smoother transitions in the tour, this would cause potential problems in the user experience and total size of the photoset. The user experience would be diminished because the user would need to click many more navigation buttons to move the same amount of distance if the pictures are closer together. Additionally since this tour is intended to be served over the internet, it is best practice to keep server bandwidth requirements modest. We used black tape because it blended in with the carpet well and thus allowed us to measure out multiple picture locations at once without the tape being too noticeable in pictures. Marking multiple locations at once also made it easier to make sure we were following a straight path as we moved the camera. On a final note for continuity, it is important to keep

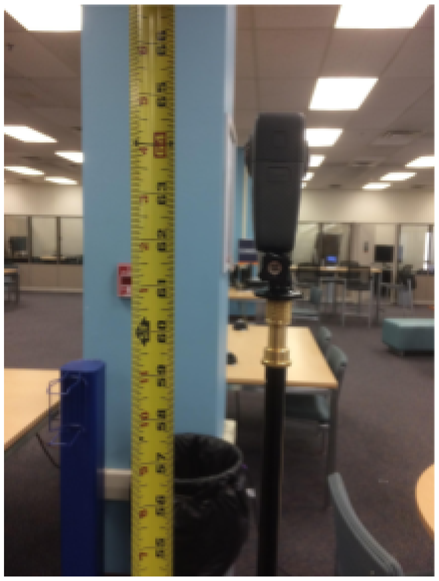


Figure 2.3

the rotation of the 360 degree camera consistent between each picture. Although the pictures are "spherical" when viewed in VR, the orientation of the camera will be the initial orientation of the user's viewport when they transition to that picture, so this initial orientation difference would need to be adjusted for in post processing. We found it was easier to do our best to keep the orientation correct while taking the pictures, that way there were only a few photos we needed to adjust in post processing.

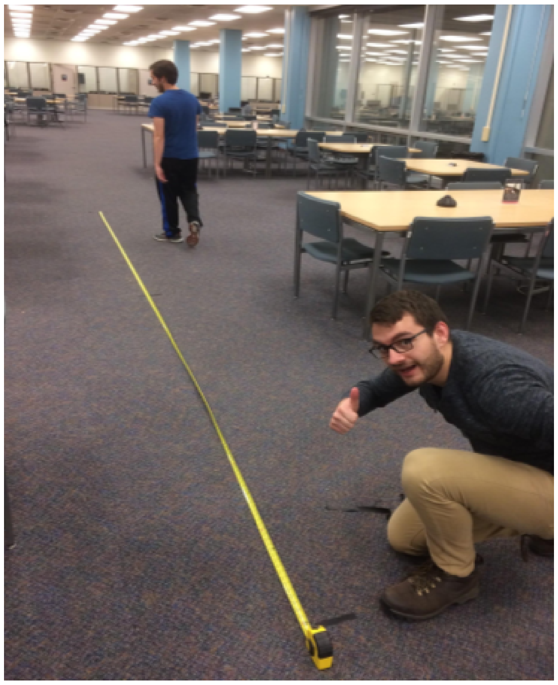
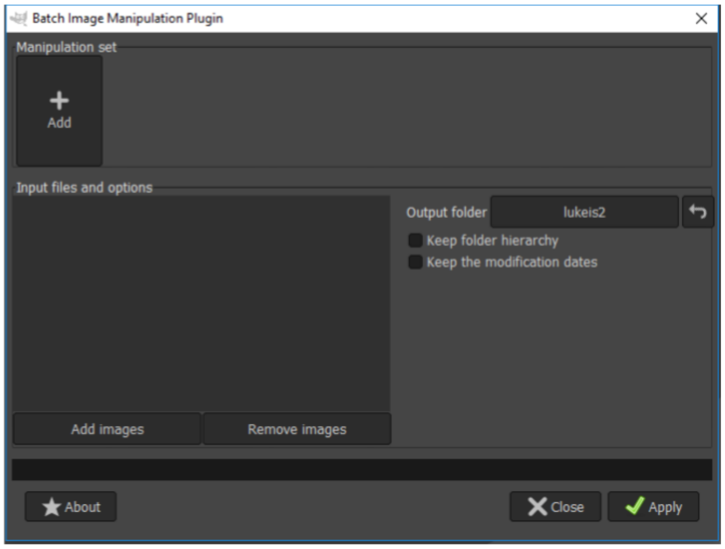


Figure 2.4

Image Processing Pipeline

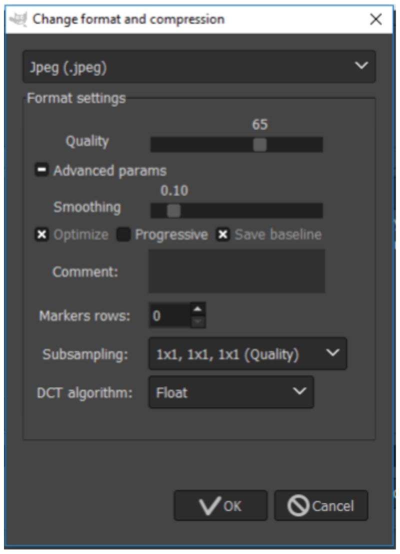
Image processing was quite simple and only consisted of using the GoPro Fusion Studio App[2] and then compressing the images using GIMP[3]. In order to import the stitched 360 degree image files, we simply connected the GoPro Fusion to a computer using the USB cable and opened the GoPro Fusion Studio App. We then selected "Browse camera media" on the screen shown when the application starts up, after several minutes (for our ~100 picture set) we were able to browse the photos in the application and select the ones we wanted by adding them to the render cue. Before adding them to the queue there are also options to adjust the orientation and exposure/color of the picture. This is where orientation corrections would be performed as needed. We did not have to adjust any other settings as the default jpeg processing performed looked fine and was consistent between pictures the set. If care was not taken to ensure consistent lighting conditions while taking the pictures, other adjustments would likely be needed to keep the images consistent. For example, it may be jarring for the user to be navigating through images with varying exposure and color temperatures, depending on the severity. Before starting the render, it is recommended to click Edit>Preferences and double check the "export location" is satisfactory.* The next step would be to click on the render que and start the export process. It is recommended to use a fairly powerful computer for this as the rendering may take awhile. We used a laptop with a recent generation quad core processor and GTX 1050 discrete graphics, which the application can take advantage of to speed up rendering.

Next, we used the BIMP (Batch Image Manipulation Plugin) for GIMP in order to easily compress the photos.** Using BIMP[4] (see figure 2.5) we first clicked "Add Images" and added the folder of rendered stitched images exported from the GoPro Fusion Studio App. Next we clicked "+Add" under the manipulation set field and chose "Change format and compression" (Figure 2.6). The parameters chosen in figure 2.6 were a result of reading the GIMP wiki[5] and experimenting with different “Quality” values. We settled on a value of 65 since this is slightly above where we started to notice distracting JPEG artifacts in some test images. After the parameters are selected, click “OK”, then click “Apply” back on the main BIMP menu to start the render process. We were able to achieve approximately a 10:1 compression ratio with only a very mild decrease in picture quality, which was only noticeable when we zoomed in and paid close attention to possible JPEG artifacts.



Output folder: lukeis2
☐ Keep folder hierarchy
☐ Keep the modification dates

Figure 2.5



Jpeg (.jpeg)
Quality: 65
Smoothing: 0.10
☒ Optimize ☐ Progressive ☒ Save baseline
Comment:
Markers rows: 0
Subsampling: 1x1, 1x1, 1x1 (Quality)
DCT algorithm: Float

Figure 2.6

*While in this menu, also take note of the “import location”. If the photos need to be loaded into the application again at a later date, the user may select “Add media”, which loads photos from disk previously imported from the camera, when the application starts. Importing from the computer's local disk will be faster in the majority of cases as modern computers tend to use fast solid state storage drives.

**We decided *not* to decrease the resolution of the images after analyzing the resolution of modern VR headsets. A HTC Vive Pro has a field of view of 110°, so at any given moment the user is looking at $\frac{110\ degrees}{360\ degrees} \approx 0.31$ of the total field of view. Given one of the highest resolution headsets on the market, the HTC Vive Pro, has a horizontal resolution of 1440 pixels and our original image is 5760 pixels wide, $5760\ pixels \times 0.31 = 1760\ pixels$ which is slightly more than our headset. Thus, decreasing the resolution more than a very small amount would significantly impact final image quality.

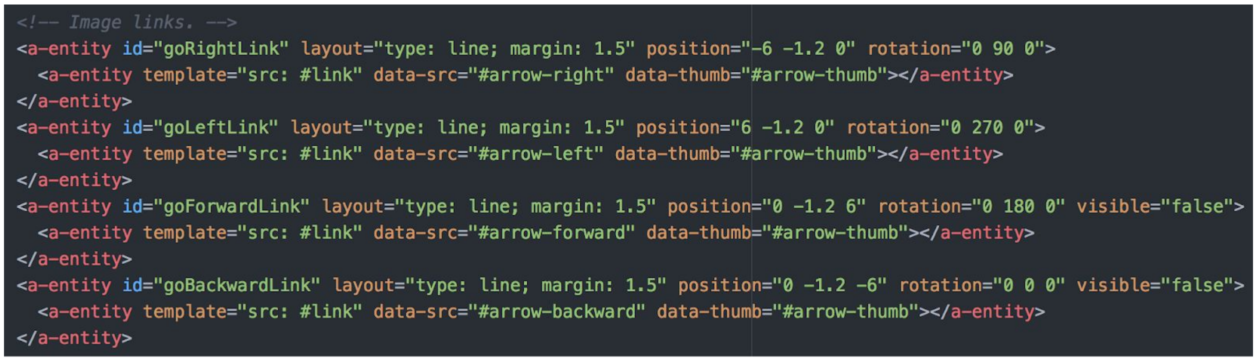
3) Web-Tour Development and Code Tutorials

Code Breakdown

The code used to create the virtual tour uses a web framework for virtual reality called A-frame[6]. The code that we used to create this program was adapted from one of the demos available on the A-frame website, specifically the demo called 360º Image Gallery[7]. Learning how to use and adapt this demo and framework allowed us to create a virtual tour of the library and has created a starting point for making it easier to preserve and create tours for other buildings in virtual reality. The code we have written consists of a folder of images along with three main files: index.html, set-image.js, and objects.js. The purpose and function of each of these will be addressed in the sections below.

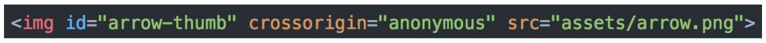
How to add and position buttons

Buttons are what allow the user to move from one location in the tour to another. Buttons can be added using the a-entity tag, which uses a template created earlier in the code to register a click. However, the code for the template will not be shown because it does not need to be modified at all. Figure 3.0 shows the creation of four buttons, with each one designed to take the user in one of the four cardinal directions. There are four factors that determine how the image appears. The first three are properties of the button. These are position, rotation, and visible. Position determines where in the space the image appears, and is defined by three numbers as *position*=”*x z y*”, where x is the position left/right, z is the position up/down, and y is the position forward/backward. Rotation determines the orientation of the button, so that it can be facing the user in a given position. Rotation is also defined by three numbers as *rotation*=”*x z y*”, where x is the rotation about the x-axis, z about the z-axis, and y about the y-axis. Thirdly, visible determines whether or not a button is visible. This is useful because not every location in a building allows for movement in all directions, so setting a button to *visible*="*false*” keeps the user from attempting to travel in that direction. Buttons are visible by default, so if you want a button to be visible you do not need to add *visible=”true”*.



```
<!-- Image links. -->
<a-entity id="goRightLink" layout="type: line; margin: 1.5" position="-6 -1.2 0" rotation="0 90 0">
  <a-entity template="src: #link" data-src="#arrow-right" data-thumb="#arrow-thumb"></a-entity>
</a-entity>
<a-entity id="goLeftLink" layout="type: line; margin: 1.5" position="6 -1.2 0" rotation="0 270 0">
  <a-entity template="src: #link" data-src="#arrow-left" data-thumb="#arrow-thumb"></a-entity>
</a-entity>
<a-entity id="goForwardLink" layout="type: line; margin: 1.5" position="0 -1.2 6" rotation="0 180 0" visible="false">
  <a-entity template="src: #link" data-src="#arrow-forward" data-thumb="#arrow-thumb"></a-entity>
</a-entity>
<a-entity id="goBackwardLink" layout="type: line; margin: 1.5" position="0 -1.2 -6" rotation="0 0 0" visible="false">
  <a-entity template="src: #link" data-src="#arrow-backward" data-thumb="#arrow-thumb"></a-entity>
</a-entity>
```

Figure 3.0



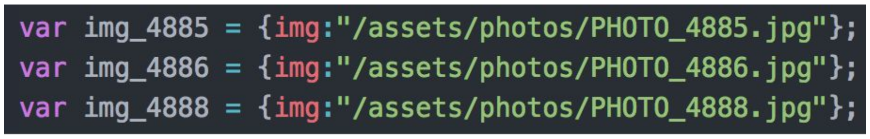
```
<img id="arrow-thumb" crossorigin="anonymous" src="assets/arrow.png">
```

Figure 3.1

The Fourth factor that determines how the button appears is the image used for the button. This is set by creating an image tag with *src* set to the location of your desired image. This image is given an *id* that is referenced by the *data-thumb* property of the button. Figure 3.1 shows creation of an image with *id="arrow-thumb"*, which is then referenced by the button in figure 3.0 by setting its *data-thumb="#arrow-thumb"*.

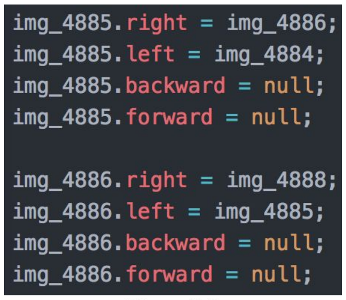
How to add images in order

For this project, we used a simple data structure to hold all of our images. This data structure is an object that has five different properties: *img, right, left, forward,* and *backward*. The *img* is assigned the location of the image, and allows the program to find and display the image. The *right, left, forward,* and *backward* properties are used as a way to navigate between pictures, determining which locations can be accessed from the current location. There is a specific order that these properties must assigned to their objects in, that is all objects must be created before any object's directional properties are set.



```
var img_4885 = {img:"/assets/photos/PHOTO_4885.jpg"};
var img_4886 = {img:"/assets/photos/PHOTO_4886.jpg"};
var img_4888 = {img:"/assets/photos/PHOTO_4888.jpg"};
```

Figure 3.2



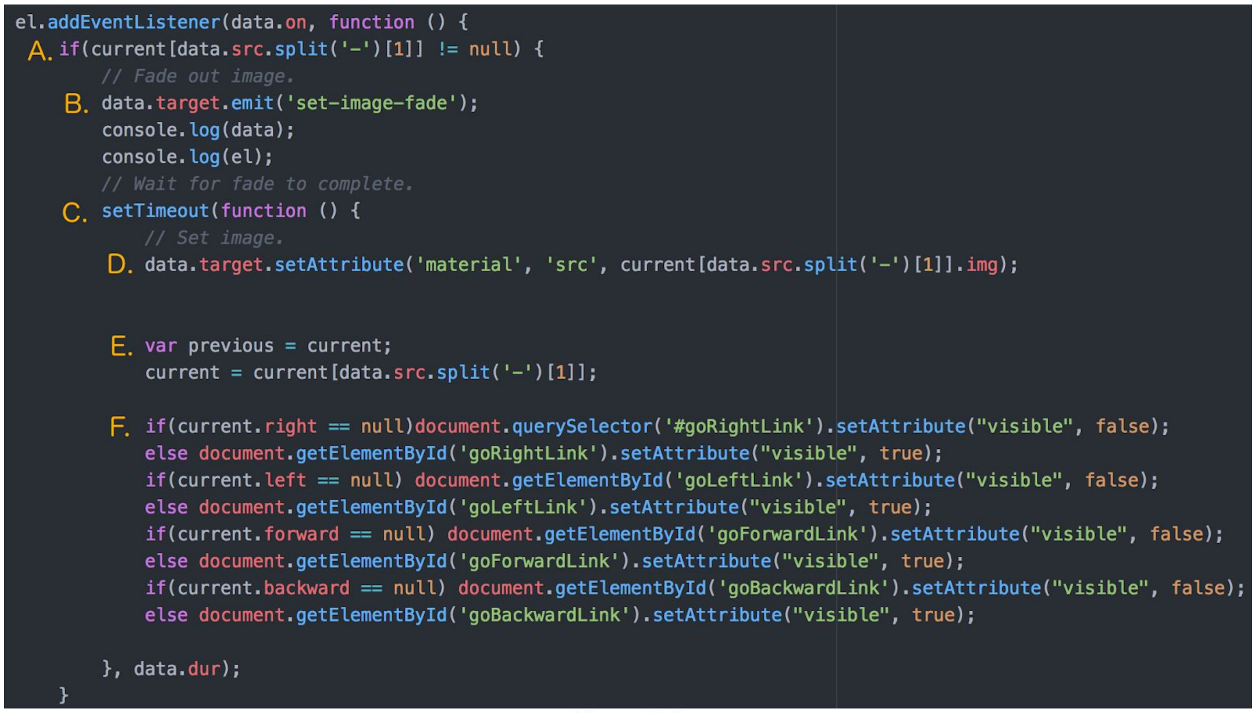
```
img_4885.right = img_4886;
img_4885.left = img_4884;
img_4885.backward = null;
img_4885.forward = null;

img_4886.right = img_4888;
img_4886.left = img_4885;
img_4886.backward = null;
img_4886.forward = null;
```

Figure 3.3

Figure 3.2 shows the creation of three objects, with each one being assigned their image sources upon creation. Figure 3.3 shows the assignment of the right, left, backward, and forward properties for each image. If a directional property is set to another object, for example *img_4885.right* in figure 3.3 is set to *img_4886*, then it means that you can get to *img_4886* by going right from *img_4885.* If a directional property is set to *null*, such as *img_4885.right* in figure 3.3, then it means that from *img_4885,* you can not travel in the *right* direction, and because of how the script is set up in the set-image.js file, the transition button will not show up for that direction. When setting up the image objects, all of the objects must be created as in figure 3.2 before setting any directional properties as in figure 3.3. The directional properties will be set based on the set of pictures that were taken. For each image taken, it is important to note which other images will be adjacent to that image, so that you can have a rather straightforward process of adding the images to the file. In figure 3.2, each object is given the name of the image that is used as its source. This is not necessary, but certainly makes it easier to keep track of which objects neighbor other objects when looking back at the images. All of the code referenced in this section is held in the objects.js file.

How to transition between images

The transition between images is set up in the index.html file and carried out in the set-image.js file. No changes to the transition would need to happen to adapt this project for use on other buildings or locations. Figure 3.4 shows the most important parts of the code that allows the image transitions, and specific parts of the code in figure 3.4 are outlined below.



```
el.addEventListener(data.on, function () {
 A. if(current[data.src.split('-')[1]] != null) {
        // Fade out image.
    B.  data.target.emit('set-image-fade');
        console.log(data);
        console.log(el);
        // Wait for fade to complete.
    C.  setTimeout(function () {
            // Set image.
        D.  data.target.setAttribute('material', 'src', current[data.src.split('-')[1]].img);


        E.  var previous = current;
            current = current[data.src.split('-')[1]];

        F.  if(current.right == null)document.querySelector('#goRightLink').setAttribute("visible", false);
            else document.getElementById('goRightLink').setAttribute("visible", true);
            if(current.left == null) document.getElementById('goLeftLink').setAttribute("visible", false);
            else document.getElementById('goLeftLink').setAttribute("visible", true);
            if(current.forward == null) document.getElementById('goForwardLink').setAttribute("visible", false);
            else document.getElementById('goForwardLink').setAttribute("visible", true);
            if(current.backward == null) document.getElementById('goBackwardLink').setAttribute("visible", false);
            else document.getElementById('goBackwardLink').setAttribute("visible", true);

        }, data.dur);
    }
```

Figure 3.4

A. In this line, *data.src.split('-')[1]* is used to get the proposed direction of transition from the button. That direction is then check from *current,* the image object that we are currently viewing. If that direction does not equal *null*, then we proceed with the transition.

B. This line starts the fade transition.

C. By setting a timeout function, there is a delay before the code within the function is executed. This is done to allow the fade transition to get to the right point before changing the image that is being presented to the viewer.

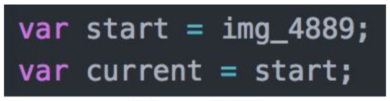
D. This line changes the image that is being presented to the viewer to the new image.

E. These two lines are what allow the transition in the data structure. The variable *previous* is used to store the current image object in case it is needed after it is changed. Then, the *current* image object is changed to the next object, which is the image object from current in the direction chosen.

F. These if/else statements check to see which directions can be travelled from the new *current* image object, hiding buttons for directions that cannot be reached and showing buttons for directions that can be reached.

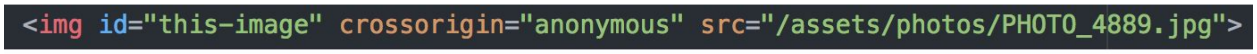
How to adapt the code for a new building/location

Using the code we have created as a base, there are few things that need to be done to adapt it to a new photoset. The main thing that needs to be done is the rewiring of the image objects in the objects.js file. All of the image objects would need to be replaced with the image objects for the new building or location, following the guidelines and instructions above. In addition, at the end of the objects.js file, two variables *start* and *current* will need to be defined, as seen in figure 3.5. *Start* indicates which of the image objects will be presented to viewers at the very beginning of the tour, and *current* will be initially be assigned to start and will control which image is being viewed at any given time. Additionally, in the index.html file the image tag with *id="this-image"* will need its *src* to be assigned to the image location of the starting image. An example of this can be seen in figure 3.6.



```
var start = img_4889;
var current = start;
```

Figure 3.5



```
<img id="this-image" crossorigin="anonymous" src="/assets/photos/PHOTO_4889.jpg">
```

Figure 3.6

Lastly, each button needs to be set as visible, if it is possible to travel in that direction from the starting image, or invisible, if it is not possible to travel in that direction from the starting image. This is done as described in the "How to add and position buttons" section of this document, and can be seen in figure 3.0.

Reference List

1. Website. Available at: https://www.amazon.com/Nikon-26513-KeyMission-360/dp/B01ASDP1SY. (Accessed: 14th December 2018)
2. Website. Available at: https://shop.gopro.com/softwareandapp/gopro-fusion-studio-app/fusion-studio.html. (Accessed: 15th December 2018)
3. GIMP. *GIMP* Available at: https://www.gimp.org/. (Accessed: 15th December 2018)
4. BIMP. Batch Image Manipulation Plugin for GIMP. | Alessandro Francesconi. Available at: https://alessandrofrancesconi.it/projects/bimp/. (Accessed: 16th December 2018)
5. GIMP/Saving as JPEG - Wikibooks, open books for an open world. Available at: https://en.wikibooks.org/wiki/GIMP/Saving_as_JPEG. (Accessed: 15th December 2018)
6. A-Frame. https://aframe.io/
7. 360º Image Gallery. https://aframe.io/examples/showcase/360-image-gallery/