Introduction

This document exists to help detail the data visualization collaboration with Jonathan Crisman and his Migrant Forensic Empathy Project this spring of 2021. Most of the work is in the form of a conversion of material from a unity version of the 3D/VR experience created by a student David (? last name ?).This document will cover choices and concessions made in the process of converting the prior material from Unity to Aframe.

Project links:

- WebVR scene: https://github.com/DevinBayly/digital_borderlands_conversion/blob/dev/src/mesh_test.
- Github Source Repo: https://github.com/DevinBayly/digital_borderlands_conversion
- Misc Storage: https://drive.google.com/drive/folders/1v42W_dgflWdtrD8QBI_OVmz3nfVLVp34?usp=sha

Background Information

Technology choices

For many people creating content that can become VR experiences there's often a big question about what framework to use. People who are creating AAA game experiences are drawn to tools like Unity and Unreal, but these don't always serve the priorities of projects that are currently at a smaller scale. For the Migrant Forensic Empathy project it made sense to go with a lighter weight framework that emphasized more rapid development at the expense of certian built in features that go with these game engines.

Technical Description Section

Landscape Model

This section will explain the work put into generating the 3D representation of a one degree square latitude/longitude section of the Arizona Border.

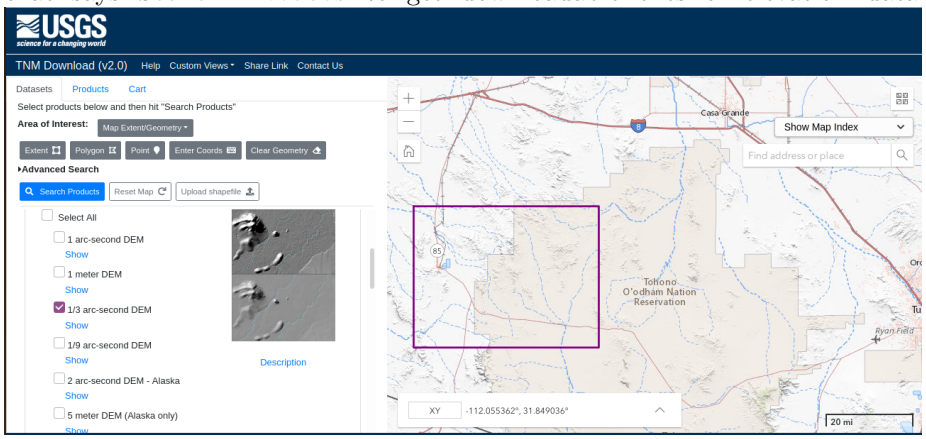
Rationale for recreating asset

There was a pre-existing landscape asset provided, but for several reasons it was more practical to regenerate the landscape for the Aframe version. Issue number one was the landscape model's file size. Model size must be decreased to deliver a decent WebVR experience on the greatest range of devices. For a single asset to be 50mb or more, this will significantly increase the start up time needed to download and load the foundation for the WebVR scene. Secondly, it was unclear what the exact latitude longitude bounding box was for the previously generated landscape. This would have prevented any accurate placement of crosses using data from the `.csv` provided by the Medical Examiner's office. Note the previous landscape used a file named Organ Pipe National monument

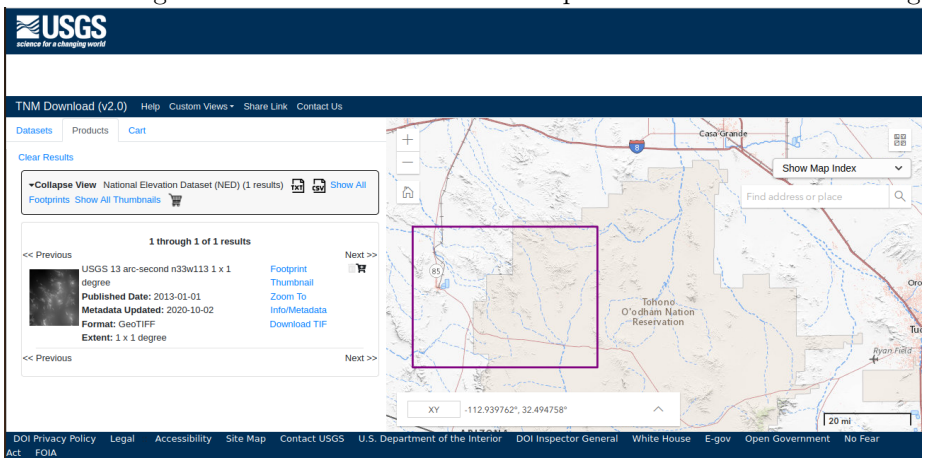
8km Height Map (USGS NED 10m).png which unfortunately wasn't enough to reverse identify the bounding box.

Acquiring Lat/Lng Bounding Box and Heightmap `.tif` Link

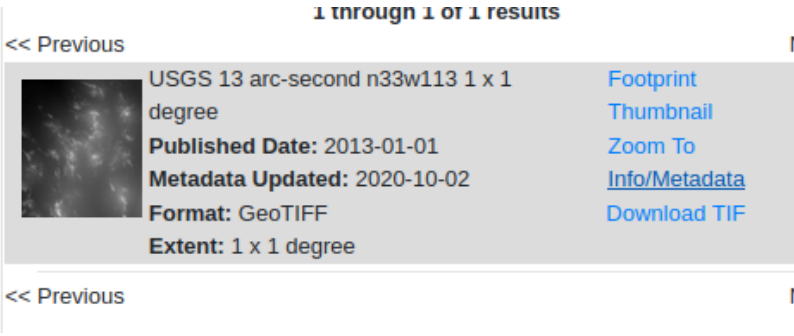
The process of generating a new landscape begins with acquiring a heightmap from USGS. USGS sites can sometimes be hard to navigate, but https://apps.nationalmap.gov/downloader/#/ and https://apps.nationalmap.gov/3depdem/ are provided here to save others time in locating elevation map browsers and downloaders. On this page click the extent button and then click and drag in the map to select area of landscape to download landscape data of whatever resolution that's needed. Then click the blue box that says `Search Products` to get downloadable tiles of elevation data.



After clicking `Search Products` the left side panel will show the following

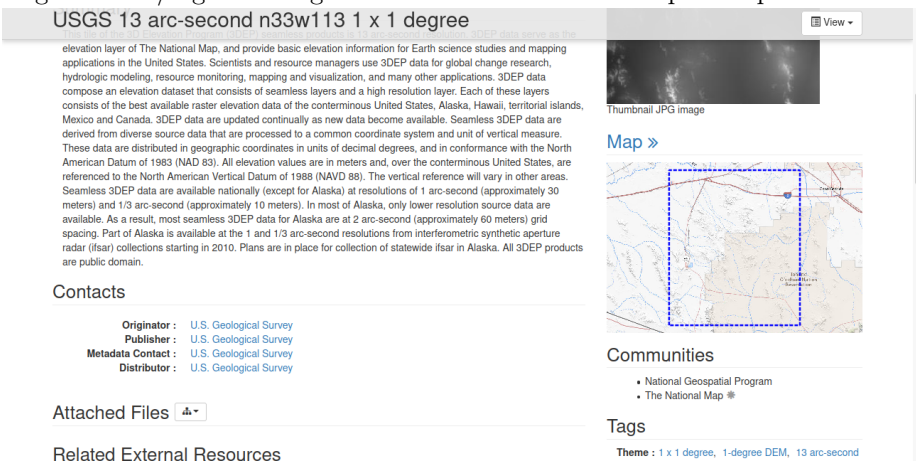


. Select the tile which bounds the region you're interested in converting into a 3D model and then click on download to get the elevation `.tif` and the

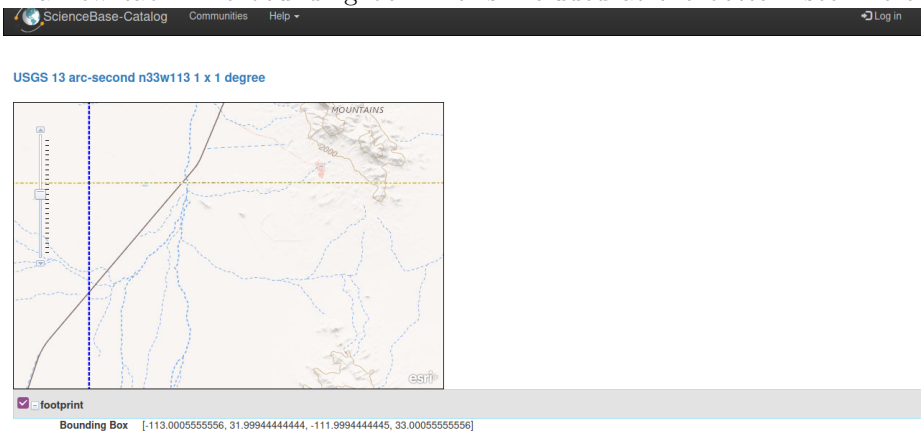


`Info/metadata` page will be used below.
It should be mentioned that the download link is more important than the `.tif` file, and will be covered in the section **Running Marching Cubes Notebook**.

It's not only the height map that is needed for the landscape conversion. In order to get the lat/lng bounding box we must click on the map and open the result



in a new tab. The bounding box info is included at the bottom seen here:



.

These values will be used in the `.csv` conversion step using the python notebook

`processing_csv_to_mesh_space.ipynb` but those detials will be included in their own section **Normalizing & Filtering Medical Examiner data**.

For the data used in this collaboration refer to the following links https://www.sciencebase.gov/catalog/item/5f7783fa82ce1d74e7d6c2c8 https://www.sciencebase.gov/catalog/item/imap/5f7783fa82ce1d74e7d6c2c8.

****Useful link for understanding how assets types correspond to physical spaces** https://www.usgs.gov/faqs/what-types-elevation-datasets-are-available-what-formats-do-they-come-and-where-can-i-download?qt-news_science_products=0#qt-news_science_products

****This section of landscape was selected because it has some overlap with the original Oregon Pipe 8km image used for the previous landscape**

Running Maching Cubes Notebook

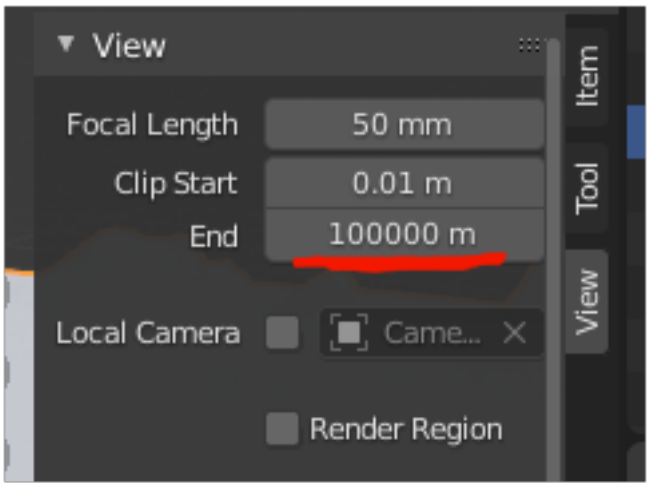
For simplicity the workflow can be re-run and modified using the binder link listed here. The idea is to provide the link to the github repository for this collaboration, and then specify the correct branch and path to the python notebook file. This will then start a jupyter-notebook on the binder servers that can be used to complete the marching cubes step running on the CPU. https://mybinder.org/v2/gh/DevinBayly/digital_borderlands_conversion/3dc5befcc144e9e6fc54b86c0419c87cae . At this link you will be able to click on the cell tab at the top and click `Run All`. It's pretty likely this will take a while so keep the tab open until the process has completed. You will know the process is done when (TODO introduce some kind of download popup or something !!).

If issues with not having enough memory for the notebook come up please use https://github.com/DevinBayly/digital_borderlands_conversion/blob/dev/python_code/actual_landscape_ma uploaded to https://colab.research.google.com/drive/1h4G2D45z-Pu5xCwBIPfjtbfCBUKxOJ-p?usp=sharing.

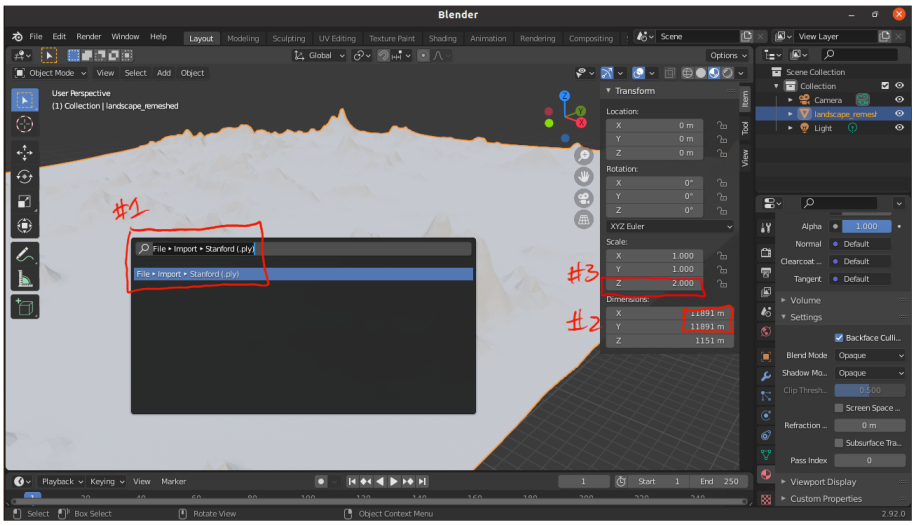
Landscape modifications

It's possible that some clean up will be necessary, and the tool chosen for this job was Meshlab (https://www.meshlab.net/). This is a complex but powerful program for manipulating point cloud data. The previous step actually used `pymeshlab` which is a Python API for Meshlab. Blender is another essential tool for working with models (https://www.blender.org/). To use the result of our computation, open Blender and use the `.ply` import option (label #1). Once you've loaded the model, nothing will appear on the screen and that's due to the scale of the model (see label #2) which is a side bar that appears when you press the `n` key. Here it's shown that the actual landscape is 11891 meters square which is the correct real world scale. The far clipping plane in blender needs to be adjusted to a much higher value to display this model, so change

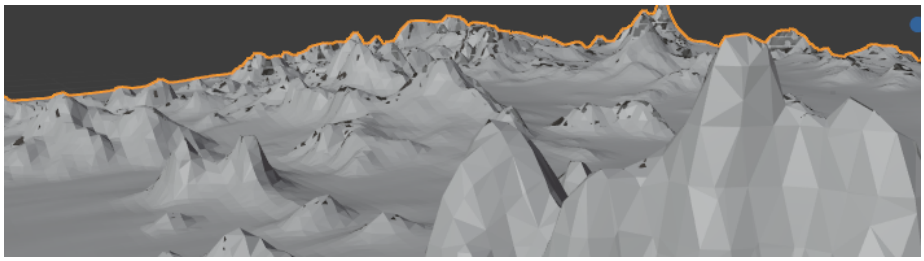
whatever number is in the `End` to 100000 meters. At this point the landscape should be visible when the zoom level is small enough.



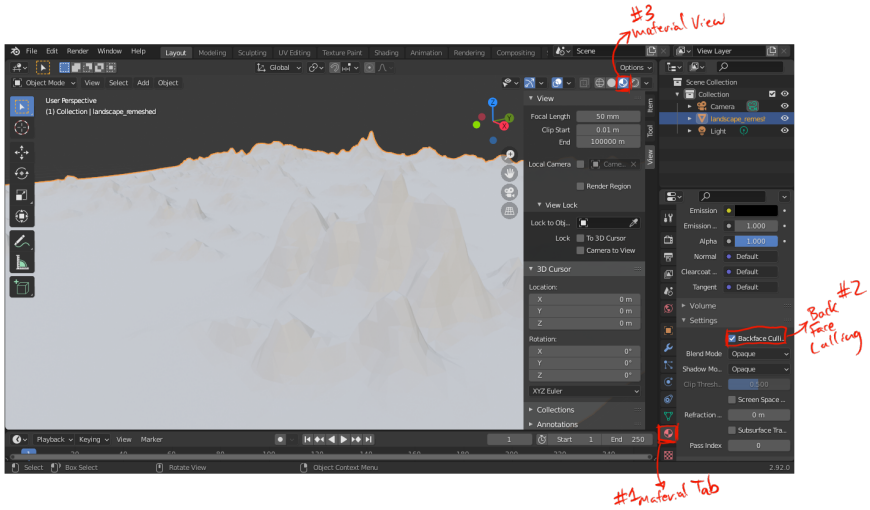
During the creation of the landscape the height values were divided by a factor of 2. This produces a much nicer landscape because the poisson-surface-reconstruction meshlab step suffers when there are sharp points in the surface. This also means that once the mesh is constructed that factor of 2 must be re-introduced (see label #3).



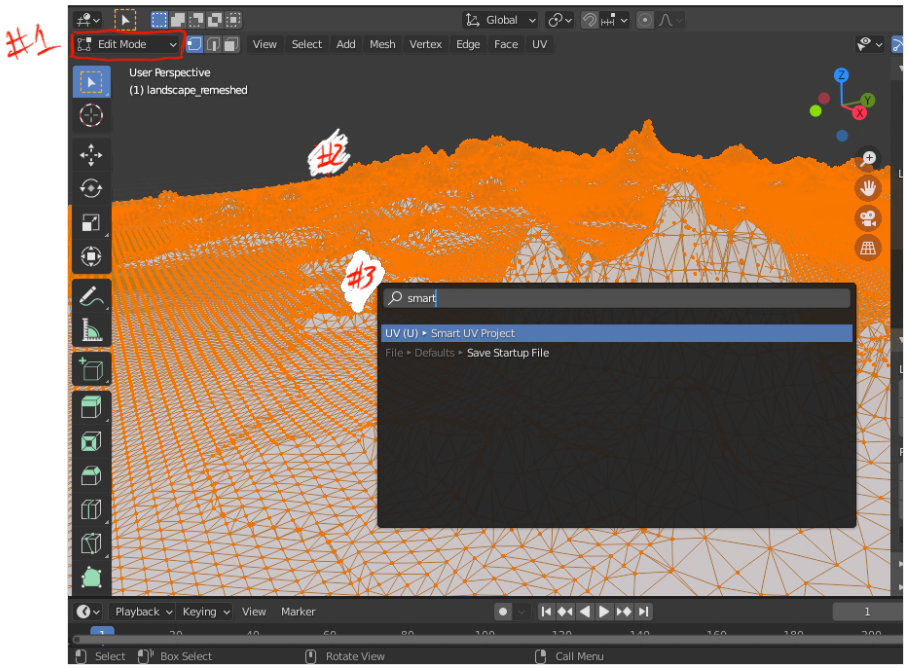
Removing artifacts is another important task in Blender. Look at the following



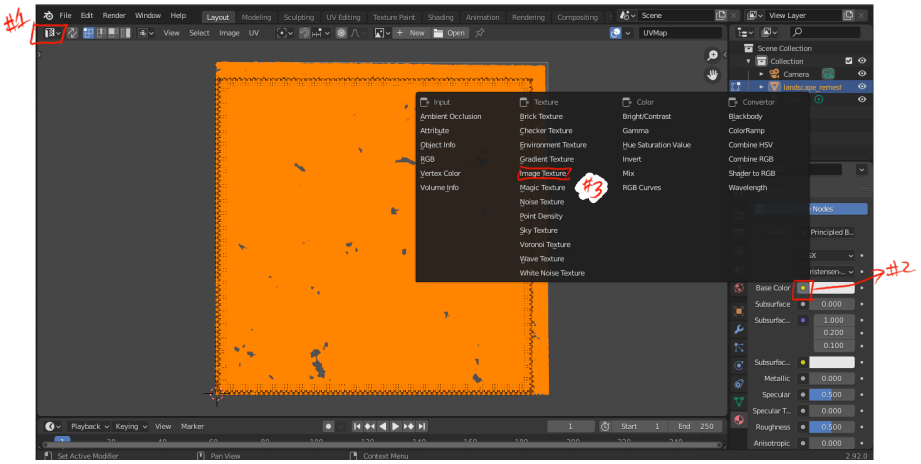
screenshot below. There are points where the surface changes height rapidly, and the shading of the mesh isn't correct. Originally it seemed that making the scale smaller was the only way to resolve this (https://stackoverflow.com/questions/67098808/large-in-meters-landscape-mesh-has-artifacts-on-peaks-only-at-certain-scale). This is actually an issue that can be tackled by clicking the material tab (label #1) setting the default material to perform whats known as `backface culling` (label #2) and then ensure that material view is on (label #3). When it comes time to export we will need to export a non-visible version of this landscape without `backface culling` in order to do landscape contstrained movement, but details about this will be provided in the **Aframe section**.



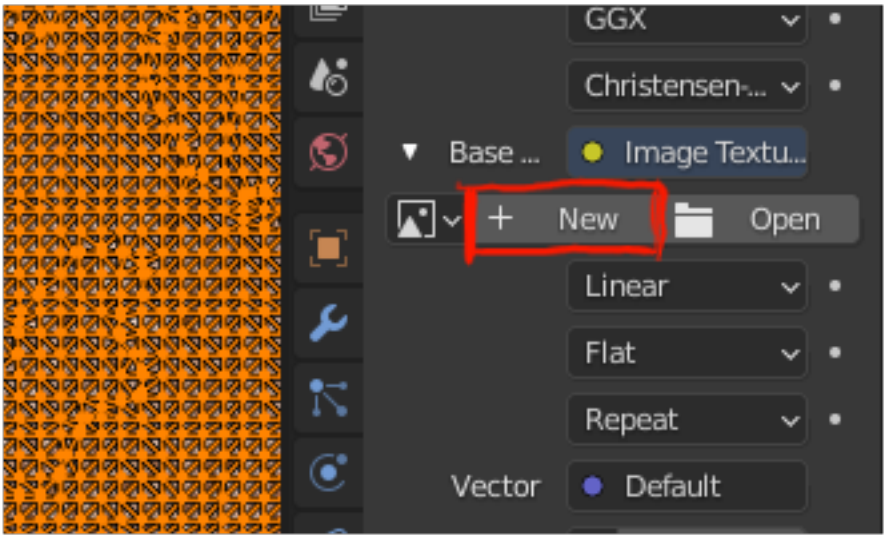
The landscape doesn't have any textures associated with it at the moment so it appears gray, there are 2 steps needed to fix that. The first is to generate UV coordinates for all our vertices. Steps here are, switch to edit mode (#1) select all vertices (ctrl-a) (#2) search and select `smart-uv project` (#3).



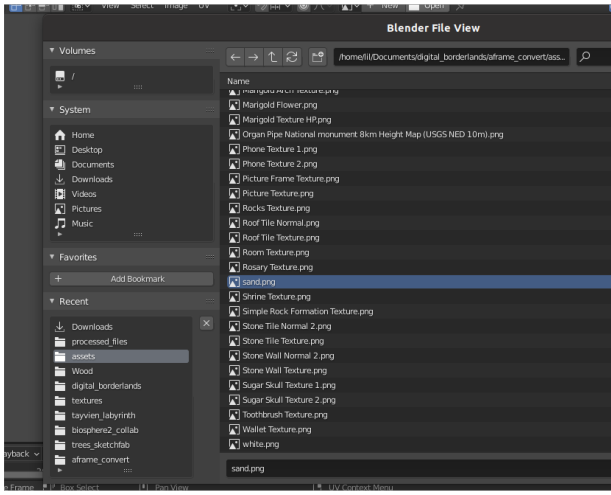
This runs and creates a UV map that we will open in the Blender UV editor. We will also choose an image to use for the mapping and that will be it.



Shown above, step 1 is select the UV editor. Step 2 is open `base color` options.

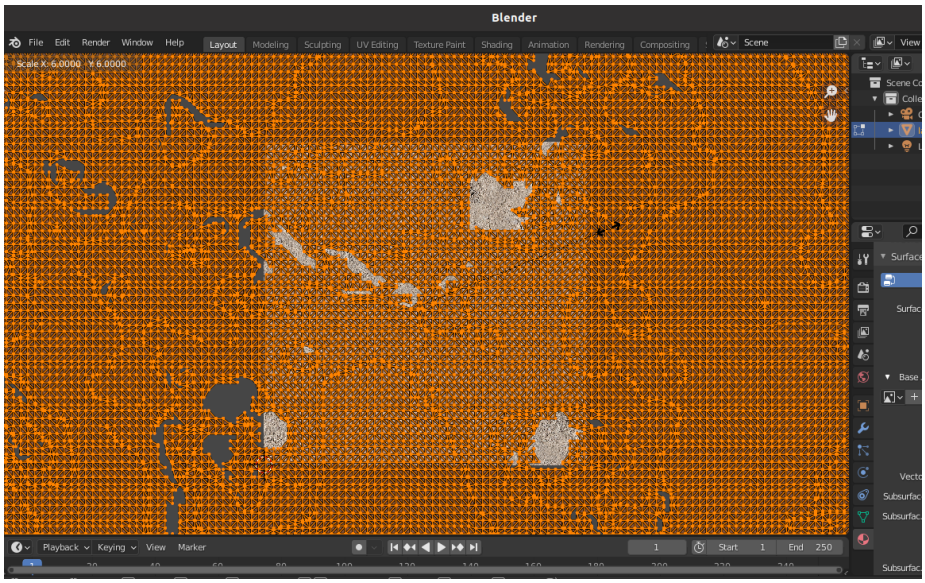


Step 3 is select `Image Texture`



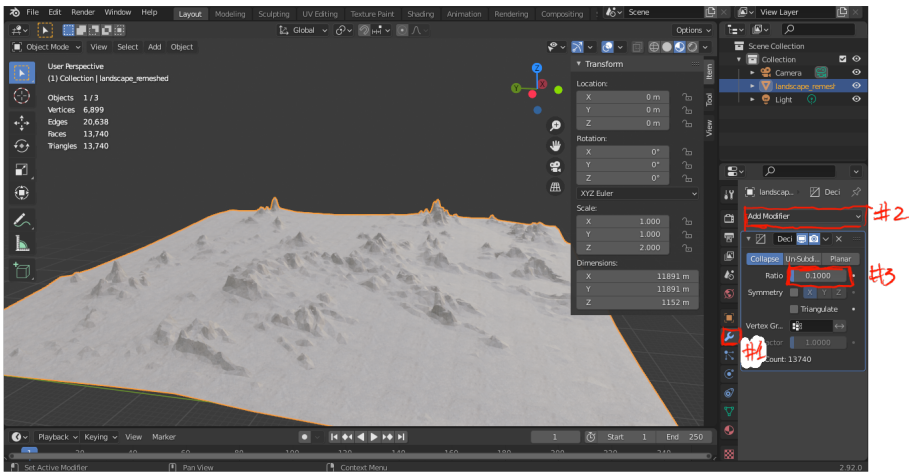
Click the `New` button and then select a `.png` texture to use.

The final part is to scale the UV coordinates up so that the texture image gets tiled across the mesh's faces. Ensure all the vertices are selected (colored orange) and press `s` and then drag the mouse towards the edge of the screen to scale up the UV coordinates.

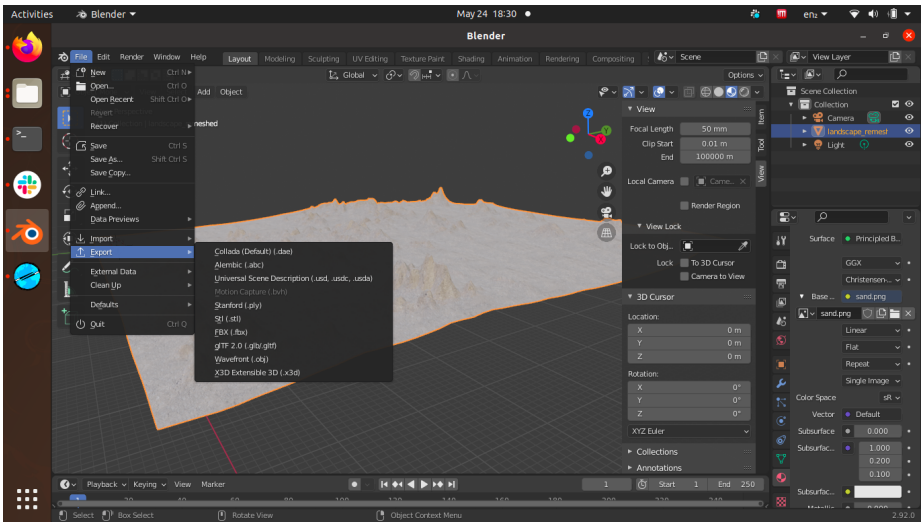


Now repeat this process until you are happy with the level of detail on your mesh's faces.

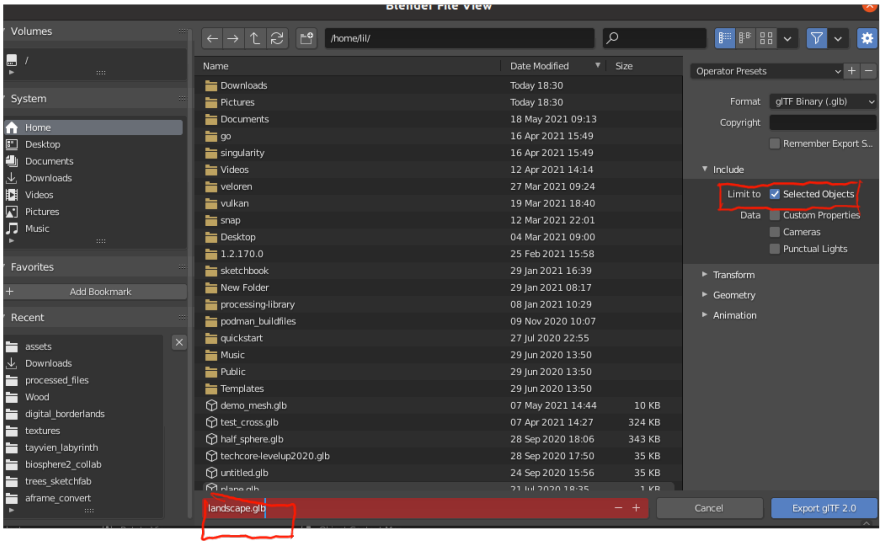
Tweaks can be made to the mesh less to improve performance. Return to 3D viewport instead of UV Editor and the Object Mode instead of Edit Mode. The easiest one is to decimate the mesh. Select the modifiers tab (#1), click dropdown options and find `decimate` (#2), then remove all but 10% of the faces (#3).



Exporting is the only final step. Click the mesh so its selected and then use `File>`



`export> gltf/glb`. The final thing to do is ensure that export is limited to selected (`Selected Objects`) and then give the exported `.glb` the name `landscape.glb`.

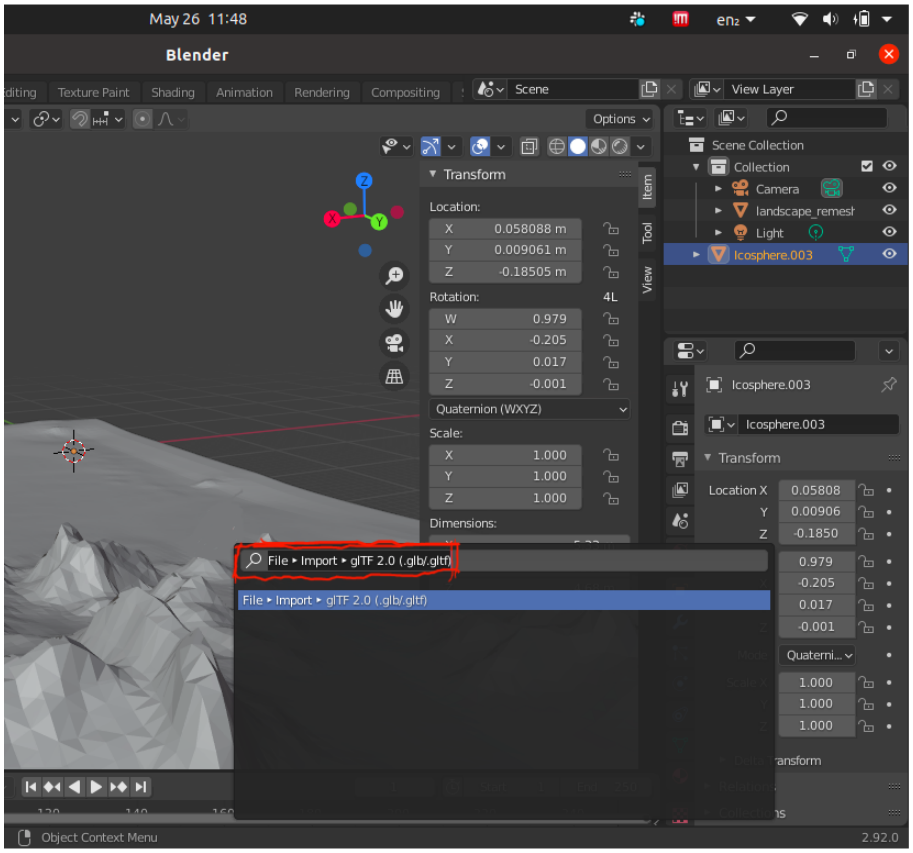


This filetype `.glb` will be the preferred type for all of the other models in our system because it allows us to use a single file with the geometry and material information contained inside of it.

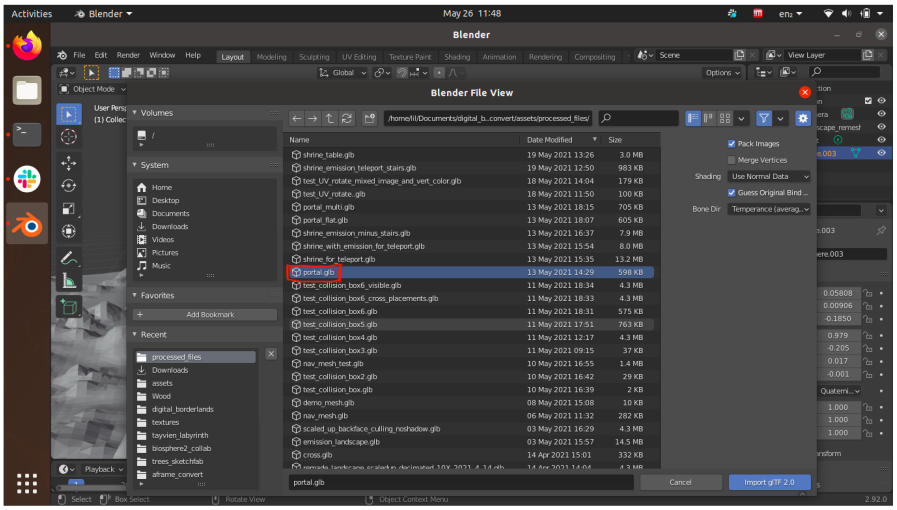
Placing other static models

As this project grows it will be important to incorporate more and more models. The approach taken for this which feels the most flexible is to perform placement and modification of models in the context of the landscape within Blender, but

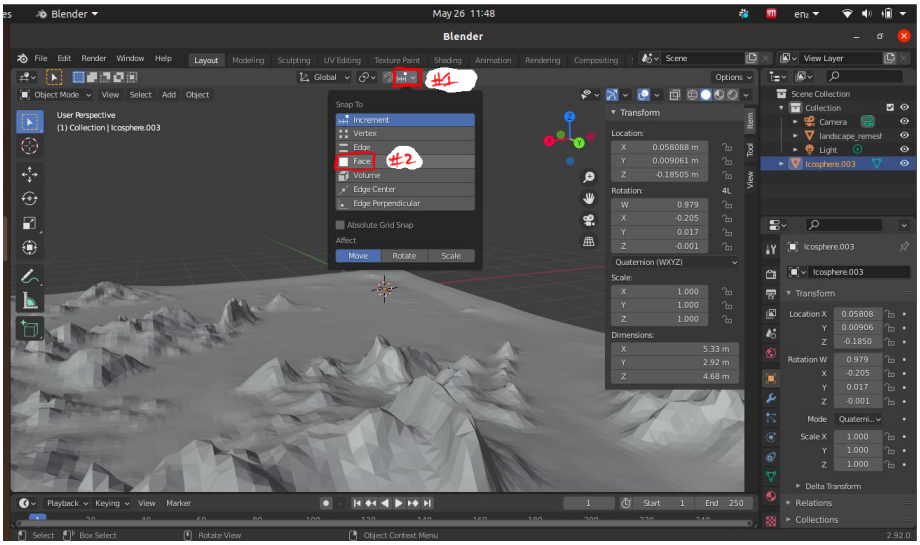
then perform separate exports. The portal placement is a concrete example of this. The following screenshots and descriptions should make it apparent how this is performed in a way that can apply to any model which needs to be placed in the context of the landscape.



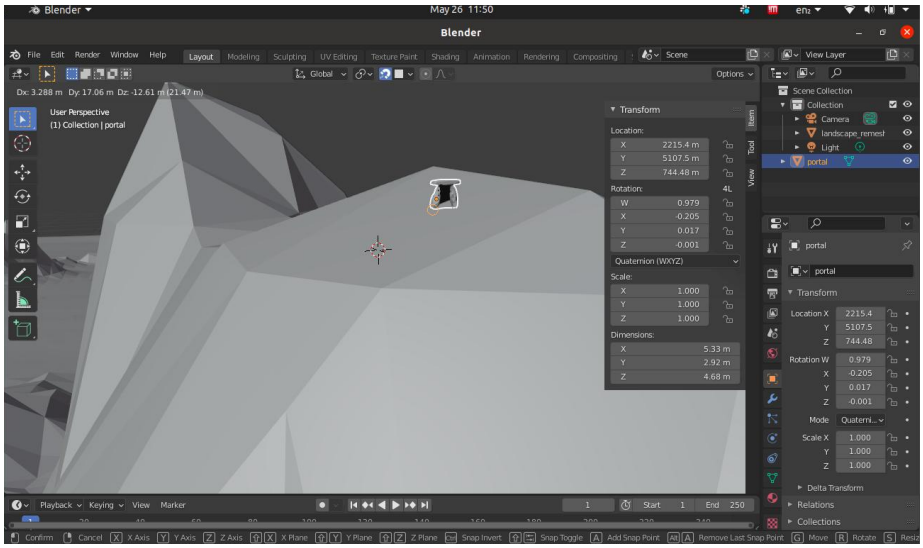
Select the gltf import option



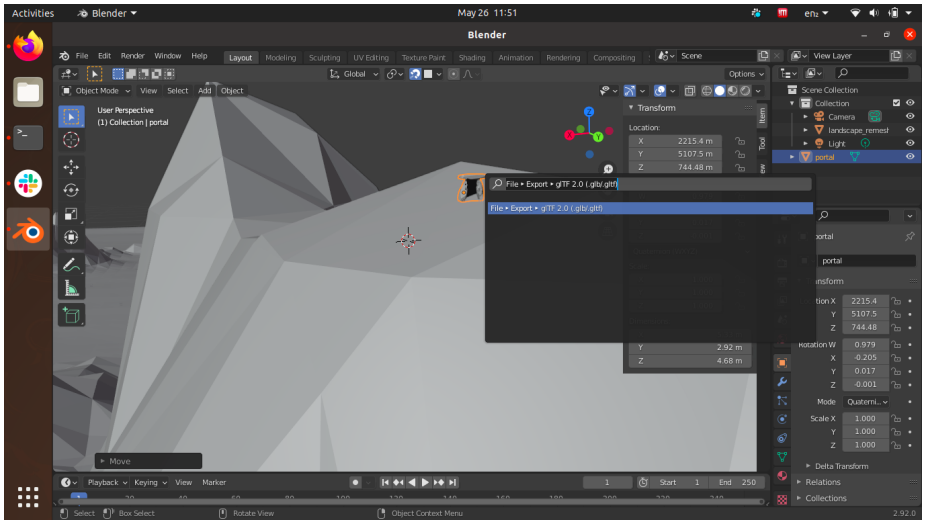
Find the model that you want to place on landscape



Select the snapping drop down (#1) and choose face snapping option in blender (#2).



Use the transform option, and when cursor is over a face the model will snap its closest part to the landscape (tip, have the model above the landscape to start)

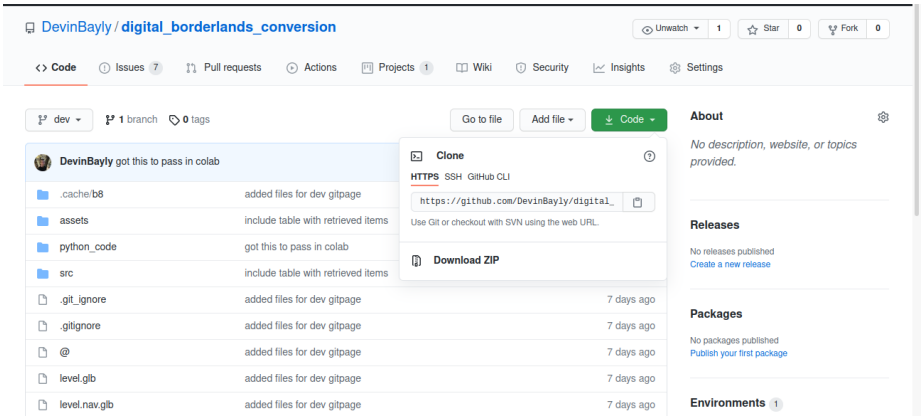


Then ensure the model is selected and export to gltf, with the `limit to selected objects` checked in the window that pops up.

This will ensure that when the model is imported into the Aframe scene, it will sit in the correct position relative to the landscape even though the landscape and the model were separate exports.

Creating the development environment

The most simple method for recreating a development environment is as follows. The first step is to navigate to the github repository and the `dev` branch https://github.com/DevinBayly/digital_borderlands_conversion. At this page the next step is to clone the repo url shown in the screenshot.



There may be some difference in how cloning happens depending on the operating system in use, but on linux open a terminal and enter `git clone https://github.com/DevinBayly/digital_borderlands_conversion.git`

Inside this new directory start a simple python3 server with `python3 -m http.server -b localhost 8080`. Then open a web browser and navigate to `http://localhost:8080/src/mesh_test.html`.

Any time that changes are made to the files inside the `src/` directory, the webbrowser page will need to be reloaded.

For more responsive reloading options check into options such as `npm reload.js` https://www.npmjs.com/package/reload.

Aframe Details

What is Aframe

It is highly encouraged that for all the material of this section the reader also consults the Aframe intro documentation at this link https://aframe.io/docs/1.2.0/introduction/.

Aframe is a Markup language similar to HTML for creating WebVR scenes. Under the hood Aframe is implemented on top of Threejs which itself is a wrapper for WebGL. This enables Aframe to tap into hardware accelerators provided by the device to render frames of the VR scene as quickly as possible. Aframe is also designed as an Entity Component System which means that many complex models used in a VR scene are actually composed of individual

components put together. In this project many aspects of the final scene are built using existing components, and only a few new ones were written from scratch. Such is a strength of Aframe that a developer can quickly build a scene reusing modular components. Another significant strength of Aframe's is the simpicity with which a developer can incorporate other web entities such as image and video elements, or even arbitrary javascript execution.

Basic Usage

A simple Aframe scene looks like the following

```
<html>
  <head>
    <script src="https://aframe.io/releases/1.2.0/aframe.min.js"></script>
  </head>
  <body>
    <a-scene>
      <a-box position="-1 0.5 -3" rotation="0 45 0" color="#4CC3D9"></a-box>
      <a-sphere position="0 1.25 -5" radius="1.25" color="#EF2D5E"></a-sphere>
      <a-cylinder position="1 0.75 -3" radius="0.5" height="1.5" color="#FFC65D"></a-cylinde
      <a-plane position="0 0 -4" rotation="-90 0 0" width="4" height="4" color="#7BC8A4"></a
      <a-sky color="#ECECEC"></a-sky>
    </a-scene>
  </body>
</html>
```

The `<a-scene>` element wraps the contents of the WebVR entities that are rendered in the viewport.

What are components?

Components are the basic building block of Aframe scenes. A component is anything like `position="0 10 0"` or `geometry=primitive:box;`. Many of the entity's like `<a-box position="-1 0.5 -3" rotation="0 45 0" color="#4CC3D9"></a-box>` are shortcuts to writing `<a-entity geometry="primitive: box; width: 3" position="-1 0.5 -3" rotation="0 45 0" material="color: #4CC3D9"></a-entity>`". Aframe comes with quite a number of solid components ready for use, but it also possible to generate them from scratch. Lines like the following are used to define a custom component in a `.js` file that is loaded before the `<body>` but after Aframe's src:

```
AFRAME.registerComponent('hello-world', {
  init: function () {
    console.log('Hello, World!');
  }
});
```

Now in the `.html` file within the `<a-scene>...</a-scene>` entities like

`<a-entity hello-world></a-entity>` can be included. This approach is used in the Aframe conversion for user interaction raycasting, loading landscape models (visible & non-visible), and setting up the machinery to place crosses near the user's camera location. This is not a topic that should be undertaken without some understanding of javascript, but many of the proposed future developments include knowing how to create new components for the Aframe scene.

details at https://aframe.io/docs/1.2.0/introduction/entity-component-system.html, and https://aframe.io/docs/1.2.0/introduction/writing-a-component.html

Importing Models

There are several options for using the models produced in Blender which are documented at https://aframe.io/docs/1.2.0/components/gltf-model.html#sidebar. This snippet is all that's necessary for the models that require no special treatment.

```
<a-scene>
  <a-assets>
    <a-asset-item id="tree" src="/path/to/tree.gltf"></a-asset-item>
  </a-assets>

  <a-entity gltf-model="#tree"></a-entity>
</a-scene>
```

If a python development server is running in the main repo folder then the `<a-asset-item> src` attribute will look like `src="/assets/processed_files/some_static_model.glb"`. Then the `<a-asset-item> id` can be used below as the source for the `gltf-model`.

Shown below is an example using real paths and filenames from the project.

```
<a-scene>
  <a-assets>
    <a-asset-item id="portal" src="/assets/processed_files/portal.glb"></a-asset-item>
  </a-assets>

  <a-entity gltf-model="#portal"></a-entity>
</a-scene>
```

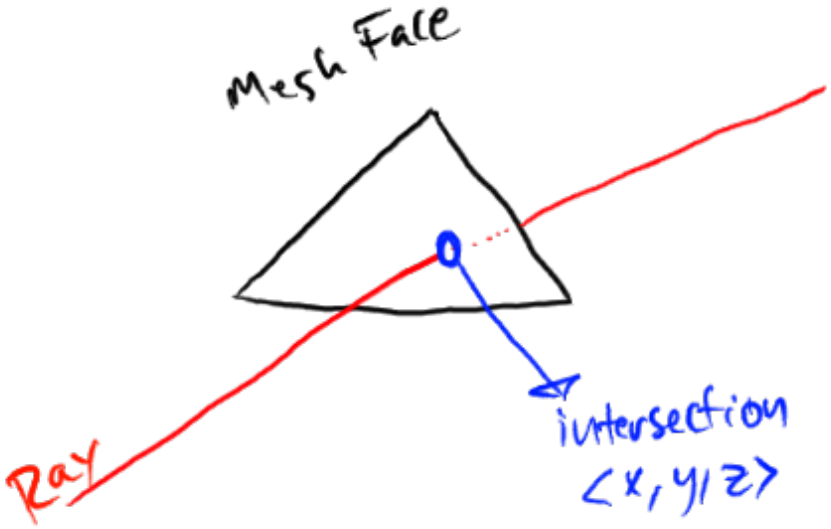
The source code contains other examples of using these imports https://github.com/DevinBayly/digital_borderla
specifically https://github.com/DevinBayly/digital_borderlands_conversion/blob/872593d5d30dd5ffe4d52ea467

Landscape Constrained Movement

The landscape must be imported before movement across its surface can be described. Many of the other game engine systems simplify the creation of

a navigation mesh that lets the player's character travel along the surface of whatever model is described as the ground. Aframe also includes such a system (https://www.donmccurdy.com/2017/08/20/creating-a-nav-mesh-for-a-webvr-scene/), but it appears to break down with models of the size used in the project. The other approach would have been to use a physics engine to calculate collisions between the user's avatar and the landscape. This becomes a prohibitively expensive operation for landscapes this size due to the frequency that collision checks must be performed for each face of the mesh. Lowering the frequency of these checks can result in miscalculations that drop the user's avatar through the mesh into the infinite void below.

These reasons provide the basis for the decision to fall back to teleportation based movement for all devices. This allows for the development of a mechanism that determines the point on the mesh that intersects with the user's intended teleport destination. Threejs has a raycaster class https://threejs.org/docs/index.html?q=raycast#api/en/core/Raycaster which is perfect for this purpose, but was only discovered after attempting to develop many custom solutions to this. The following picture will help explain what raycasting does.



The cursor component https://aframe.io/docs/1.2.0/components/cursor.html#sidebar, https://aframe.io/docs/1.2.0/components/cursor.html#intersection-data provides a mechanism to track where the user clicks on the screen, and if that click *intersects* with the mesh at some location on the mesh. The intersection point can then be used as the updated position of the camera thereby simulating a teleport. This also ensures that the user never ends up passing through the mesh which ruins the impression of solidity.

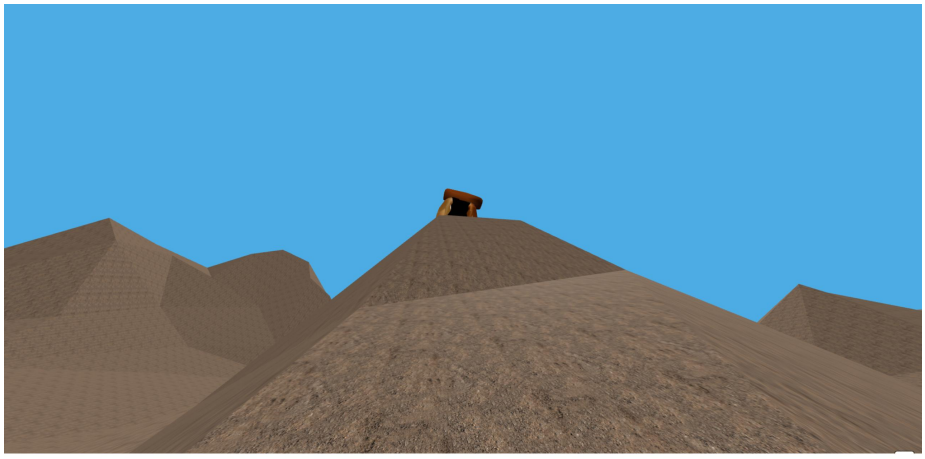
The small hitch is that a doublesided mesh is required in order to raycast with threejs and Aframe. This means that backface culling cannot be in effect. The work around for this is a little bit complex, and requires 2 landscape exports from Blender. The export process described in the landscape modifications section is used for the visible backface culled landscape. However, the raycaster landscape must be exported with the backface culling material option turned off and the ground texture removed. The `visible=false` attribute is also set on the `<a-entity>` that holds the raycasted landscape to ensure its non textured surface isn't seen by the user. The good news is this visibility setting doesn't prevent the raycasting from calculating intersections and teleporting the camera when the user clicks to a new location.

The section of the code responsible for handling this behavior is (https://github.com/DevinBayly/digital_borderla
Again much of the actual specifics are similar to the Aframe documentation for raycaster (https://aframe.io/docs/1.2.0/components/raycaster.html#example) and detecting user's cursor behavior.

Note: Currently the mouse is also used for dragging to change the rotation of the camera so to distinguish the `click` that starts a `drag` from the `click` that represents the user's intent to teleport the `shift` key must be held down for a landscape teleportation to occur. This is a mechanism that should be updated when porting to other devices and the section of code responsible for it is at https://github.com/DevinBayly/digital_borderlands_conversion/blob/ee5308a6b5260a15cf906bdbe5171c9c7eeb

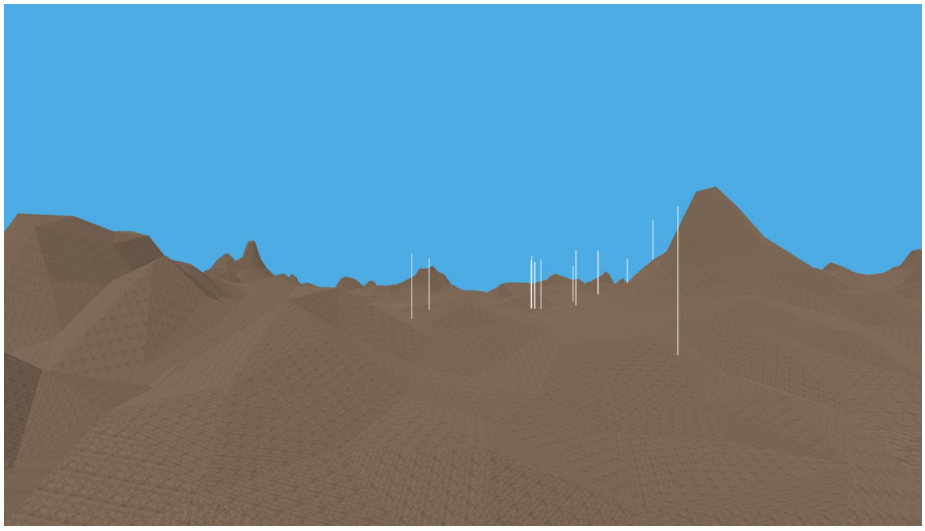
Tweaking Landscape Aesthetics

The visible landscape model comes with a texture that is mapped to the faces of the mesh, but it is far too regular and the eye can pick up the patterns which hurts the immersion in the scene. Notice that each face has the same pattern of light and dark parts arranged.



Instead a small rotation can be made to the `UV` coordinates that

map pixels of the image texture onto the mesh face. This isn't a complete fix for the issue, but it is a step in the right direction (https://github.com/DevinBayly/digital_borderlands_conversion/blob/ee5308a6b5260a15cf906bdbe5171c9c7eeb
The result is that the texture is no longer arrayed in the same direction for every face which adds some variety to the landscape.



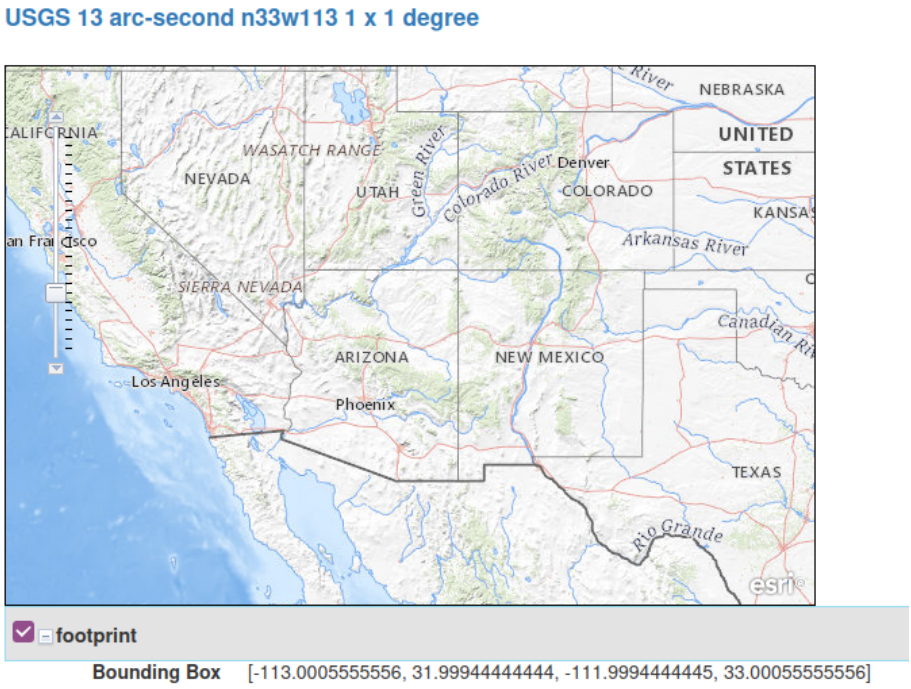
Crosses and Processing Medical Examiner Data

Normalizing and Filtering

This section will provide details on the Medical Examiner data pre-processing step as well as the placement algorithm utilized during the user's WebVR session. The data that was provided is in the form of a `.csv` (https://github.com/DevinBayly/digital_borderlands_conversion/blob/dev/python_code/ogis_migrant_deaths
This file contains columns

```
[(0, 'ML Number'),
 (1, 'Name'),
 (2, 'Sex'),
 (3, 'Age'),
 (4, 'Reporting Date'),
 (5, 'Surface Management'),
 (6, 'Location'),
 (7, 'Location Precision'),
 (8, 'Corridor Code'),
 (9, 'Corridor'),
 (10, 'Cause of Death'),
 (11, 'OME Determined COD'),
 (12, 'Condition Code'),
 (13, 'Body Condition'),
 (14, 'Post Mortem Interval'),
 (15, 'State'),
 (16, 'County'),
 (17, 'Latitude'),
 (18, 'Longitude'),
 (19, 'UTM X'),
 (20, 'UTM Y'),
 (21, 'norm_lat'),
 (22, 'norm_lng')]
```

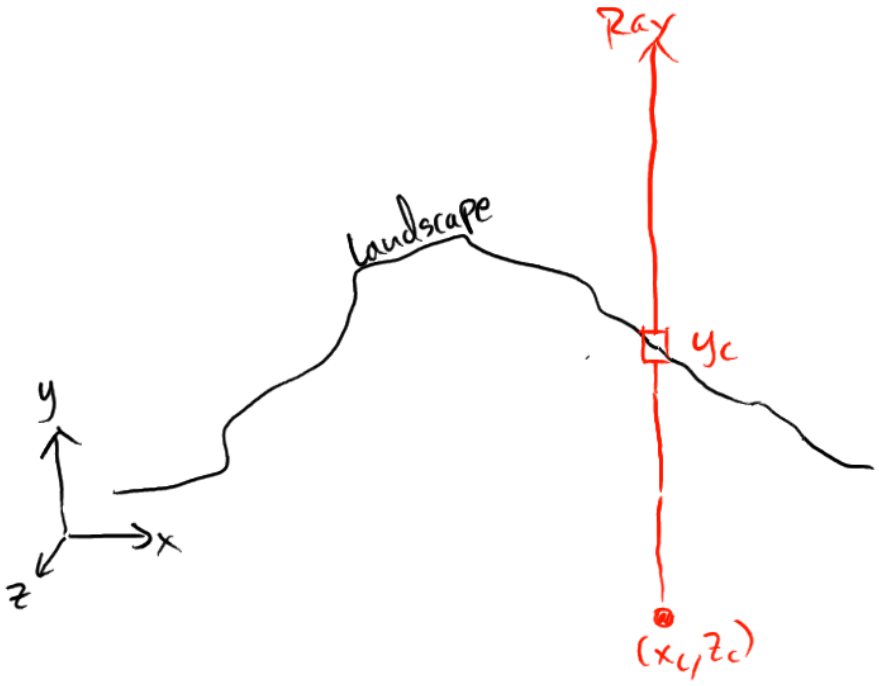
and the `norm_lat` `norm_lng` are created by running the following jupyter notebook (https://github.com/DevinBayly/digital_borderlands_conversion/blob/dev/python_code/processing_csv
This notebook first reads the `.csv` into a python Pandas dataframe. From the dataframe we use the **bounding box** information that is available for our section of landscape here https://www.sciencebase.gov/catalog/item/imap/5f7783fa82ce1d74e7d6c2c8



This bounding Box information is used for filtering the `.csv` data in columns 17 and 18. Any of the rows that don't correspond to a migrant death within the bounding box are filtered out. The next step is to normalize the values within the bounding box so that they are now in the range of `[0-1]` for lat/lng. This is critical so that later these values can be multiplied by the mesh dimensions (18911 meters) and the cross position in terms of `x,z` meters (note: `y` is up or down in the Aframe scene).

Placing Crosses

The raycasting mechanism is used for placing crosses on the landscape as well. As mentioned above, it is possible to multiply the crosses' normalized position by the landscape dimensions to arrive at the location in `x,z` that the cross should be placed, but how to determine the `y`, or height that the cross needs to be? This is where the raycasting is used. Please look at the following figure for illustration of the process thinking of a narrow slice of the landscape taken at only one `z` value



Imagine that `x_c,z_c` are the crosses `x,z` positions and the start of the ray from below the landscape. Threejs casts a ray in the positive `y` direction and calculates the `y_c` position which is the precise spot where the ray intersects the landscape. Recall just briefly that this isn't the visible landscape that is being checked for ray intersection, but the doublesided non-visible landscape that is used in the Aframe scene for raycasting in specific.

This process is repeated for each cross that is within `cross_distance_max` (https://github.com/DevinBayly/digital_borderlands_conversion/blob/ee5308a6b5260a15cf906bdbe5171c9c7eeb of the camera.

Cross Interaction

Another type of ray casting is used to determine when a user has clicked on a cross (https://github.com/DevinBayly/digital_borderlands_conversion/blob/ee5308a6b5260a15cf906bdbe5171c9c7eeb This section lets the scene know a user has clicked on a cross, and will create floating text in the air above the cross listing information about the migrant death that the cross represents.

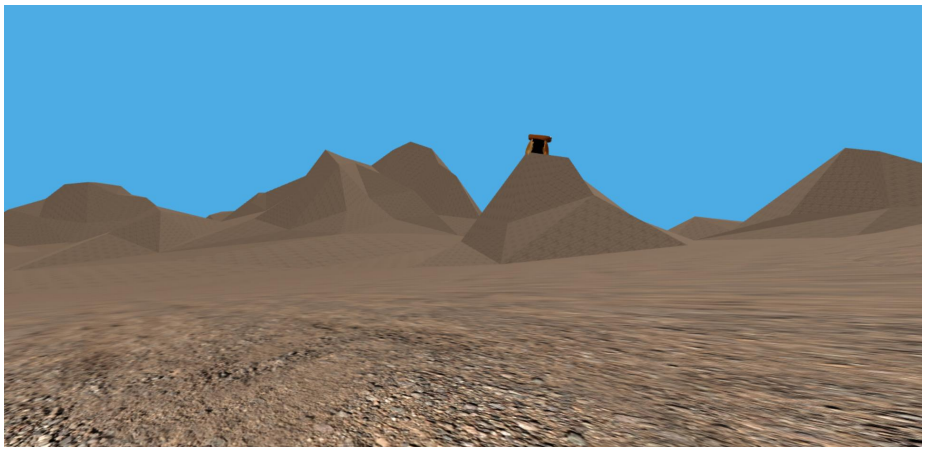
Cross Generation and Removal

As of the writing of this document there are 671 crosses which can be placed. The operation for placing the cross is almost certainly not as performant as it could

be so certain steps must be taken to not reduce the performance of the overall scene. One of the steps taken is to make the placement an asynchronous task (https://github.com/DevinBayly/digital_borderlands_conversion/blob/ee5308a6b5260a15cf906bdbe5171c9c7eeb
This code makes a slight timeout associated with each placement task, and wraps the placement in a JS promise so that it will only be handled when the machine isn't doing more important tasks. Critical to the placement and the removal is the determination of distance from the user's camera. When the camera is too far from an actual cross it will be removed from the active list of placed crosses. The cross representation list is (https://github.com/DevinBayly/digital_borderlands_conversion/blob/ee5308a6b5260a15cf906bdbe5171c9c7eeb
and is modified by placement and removal. The overhead of the scene is much less at this point with the tradeoff that crosses will take time to fully populate around the user. The assumption is that with the limitation on travel speed and the overall scale of the landscape the user won't "miss" a cross before they've left a certain area.

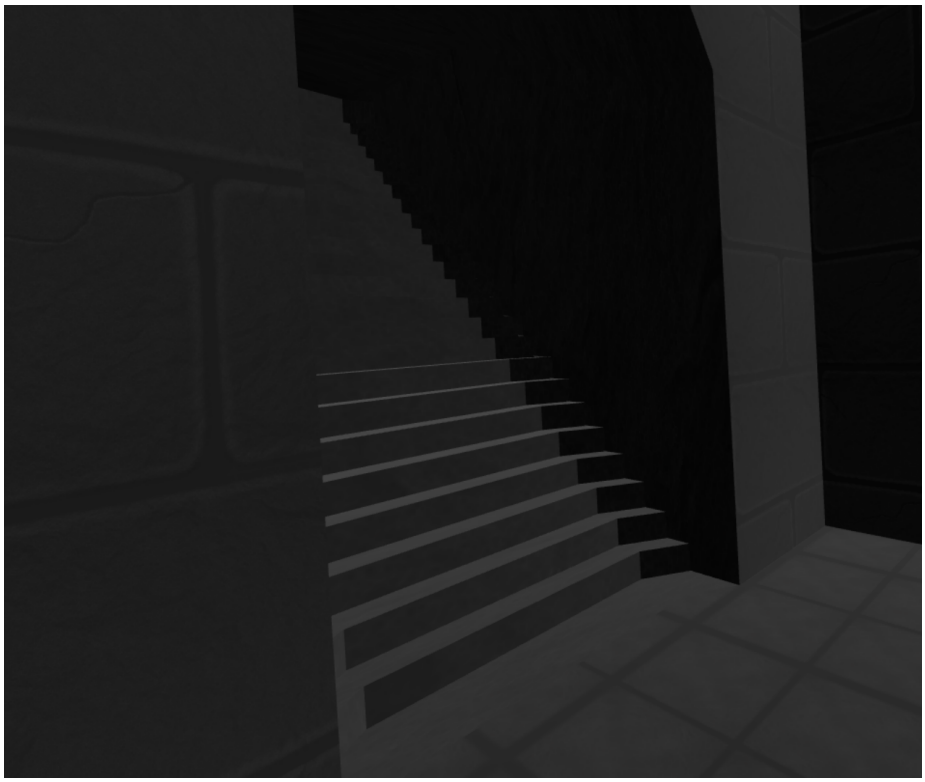
Shrine Transport

Effectively the Aframe scene can be thought of in 2 parts: the Landscape and the Shrine. Outside in the landscape there are crosses to visit and a wide open space to explore. The Shrine is a model created by previous the developer and is the home of the Migrant Artifacts. This is also the space likely to see the most development over time, but how does a user get from one to the other? The same `click` event listener is used to detect when a user is mousing over the **portal** (https://github.com/DevinBayly/digital_borderlands_conversion/blob/ee5308a6b5260a15cf906bdbe5171c9c7eeb



One end of the room has a table with various 3D models of migrant artifacts (wallet, toothbrush, cellphone, comb. . .)

and the other end has stairs that will take the user back to the landscape when clicked (https://github.com/DevinBayly/digital_borderlands_conversion/blob/ee5308a6b5260a15cf906bdbe5171



The actual model for the portal is arbitrary and should perhaps be the first thing replaced by those who continue on with this project. When the user clicks on the portal an event is generated that changes the coordinates of the camera to be the location of the shrine (physically located underneath the landscape in open space). The stairs from the shrine have been separated from the rest of the shrine model and exported from Blender on their own. The reason for this is it makes it possible to attach a different event listener to that section of the model compared with the floor of the shrine. As more development occurs within the Shrine keep this principle in mind: new parts of the shrine can be connected to the existing, but don't have to be part of the same export to be **placed** correctly relative to the rest of the shrine. To explain this more, think about how the stairs in the above image **appear** in the correct location to look connected with the rest of the shrine. This is still possible even though the stairs are part of a different export. This is a modular design approach that should provide great flexibility for how the shrine can grow over time from many different individual contributions.

Time Sinks and Lessons Learned

Like most projects, much of the development of this project isn't reflected in the final product. The biggest overall time sink was saved by the discovery of the Threejs RayCaster which is used extensively in this project now. Take home lesson is that looking at the examples presented on the Threejs page would have saved a great deal of time https://threejs.org/examples/?q=ray#webgl_geometry_terrain_raycast.

Cross Placements using quadtrees

Quite a few approaches were used to tackle the issues of intersection with the mesh. This phenomena comes up when attempting to place crosses, as well as calculating the position a camera should be so that it remains constrained to the landscape mesh. All of the attempts to optimize the search for vertices of the landscape closest to the user camera or the crosses `x,z` position were ultimately less performant than using Threejs Raycaster. For those curious about the previous aproaches used feel free to look at https://github.com/DevinBayly/qt_ll which is a rust repo for creating and querying a quadtree from vertices in a mesh `.obj`.

Landscape Reconstructions

Another major place that time went was in the creation of the landscape mesh. Before reaching the much more efficient method laid out in https://colab.research.google.com/drive/1h4G2D45z-Pu5xCwBIPfjtbfCBUKxOJp?usp=sharing. Previously the marching cubes algorithm from python scikit was used. The issue here came from needing a large amount of memory to reconstruct even a small part of the landscape so there was a good deal of time spent partitioning the landscape into smaller chunks that each could be fed to the marching cubes. The result was a process that took almost an hour to run compared to the several minutes needed to run the google colab.

Limitations

- don't currently support vr headsets
- Isn't built to handle mobile events (look into duplicating `click` as `touch` for smart phones)
- Crosses are placed on a rolling low priority basis to not reduce overall performance of scene, **could** potentially mean a user misses a cross in their vicinity, smaller `crossDistanceMax` can help with this.
- No physics for interactions.

Ongoing Development

Modifying Models

- Description: Many of the models in use aren't in their final form, or perhaps will never have a final form, but will continue to change reflecting the dynamic nature of the Migrant border experience. The shrine for instance will grow and include new wings with new material. Another clear example will be substituting the existing portal for a different model.
- Suggested Skills: Blender, basic understanding of importing models into Aframe scenes.

photo shop color correction of the landscape

- Description: Currently the landscape isn't the most aesthetic pleasing that it could be. For instance, even though the `sand.png` appears like it will be the right hue, the actual textured landscape appears very brown. This development could be to use a photo editing program to create a more *desert* like landscape texture. As long as the file is included as the texture image in Blender when the landscape is exported the rest of the Aframe scene pipeline won't have to be changed in order to render the new landscape.
- Suggested Skills: Photoshop/Gimp, Blender

creating trails in the desert

- Description: Jonathan Crisman asked whether the white cube that is placed upon teleport could be swapped out for something more appropriate like a pair of footprints or shoe prints. This task would require creating a model that should be placed, and then understanding how to update the src to load a model and place it at the appropriate location. The edits will be in the vicinity of this code https://github.com/DevinBayly/digital_borderlands_conversion/blob/ee5308a6b5260a15cf906bdbe5171c9
- Suggested Skills: minor Javascript coding, Blender, Aframe basic knowledge

having a mechanism to connect to a google maps esque viewer

- Description: This is one of the more complex directions that the project can go in. The idea would be to have an actual google maps window located in the bottom right. All of the same types of interaction that users are accustomed to would be included, but there would be a way to indicate an intent to teleport to a particular location on the minimap with the Aframe scene Camera.
- Suggested Skills: Advanced Javascript, Experience using web API's, Aframe intermediate understanding

github action supported landscape conversion via prebuild docker container

- Description: This task would be to convert the code in the google colab into a script that runs within a docker container that can be triggered to start by a github action. What's nice about this is it could be used to automatically re-run the landscape conversion for free provided a url of a landscape DEM `.tif` as long as the process doesn't take more than 6 hours.
- Suggested Skills: Docker containerization, python scripting, github action YAML

limited travel distance for teleportation ~10m

- Description: This should be a fairly straightforward task. It appears that the near and far properties of the raycaster component can be tweaked to create a limit on the motion amount that fits with the intended experiential quality of movement https://aframe.io/docs/1.2.0/components/raycaster.html#properties .
- Suggested Skills: Aframe basic knowledge

rotate text to face way the cross is

- Description: When text appears above a cross it is almost always not pointing the right direction. It would be amazing to have it face the user wherever they were when they clicked the cross, but the simpler approach suggested by Jonathan was to have the text atleast always line up with the orientation of the face of the cross.
- Suggested Skills: beginner level Javascript, Aframe basic understanding

alternating transparency for cross beacons

- Description: Currently the markers for a cross are just white lines that stick up in the `y` direction. Jonathan requested that a different aesthetic be used for these beacons, perhaps that feels more ephemeral than the solid white line. It might be easiest to have the transparency of the lines cycle over time so that they appear to waver.
- Suggested Skills: intermediate Javascript/understanding setTimeout or setInterval, Aframe basic understanding

correct coloring on the portal

- Description: The current portal has a baked texture map that is pretty incongruous with the rest of the landscape, so it is desirable to change the texture of the model. Once the new texture is created the full portal model should be exported from Blender with the same name and file path. This

will ensure that the rest of the WebVR Aframe pipeline won't have to be changed in order to show the changes to the portal texture in the scene.
- Suggested Skills: Blender

switch portal to less pagan looking model

- Description: The current portal is a bit too stone henge like, so this task is to use Blender to design a new portal. The idea would be to load the full landscape and position the new portal accordingly before exporting with the name `portal.glb` so that the rest of the Aframe WebVR pipeline doesn't have to be updated to reflect the changes in the scene.
- Suggested Skills: Blender

adding physics to smaller spaces (shrine)

- Description: Physics is a tricky subject so this task should only be attempted by someone familiar with Javascript, and Aframe. The physics library that appears to be most in use is https://github.com/n5ro/aframe-physics-system, but that it requires pretty small meshes to function. This means that instead of trying to put physics on the whole landscape, it should only be added to the shrine. This would mean the shrine is the static mesh, and the objects within the shrine that are supposed to be dynamic would collide with the floor/table/walls/ceiling.
- Suggested Skills: intermediate Javascript and Aframe

putting in the oral histories

- Description: This task is meant to increase the variety of asset present in the shrine by include a 2d plane connected to a youtube video of recorded oral histories of migrants crossing the border. The Aframe example that is closest to this is https://aframe.io/aframe/examples/test/video/.
- Suggested Skills: intermediate Javascript and Aframe

Templating out the messages that appear above the cross

- Description: The data contained in the `.csv` isn't in a very user friendly format so this task is aimed at creating a sentence template that can be filled with bits of specific data which makes in more readable. This will require updating the following section of code https://github.com/DevinBayly/digital_borderlands_conversion/blob/ee5308a6b5260a15cf906bdbe5171c9
.
- Suggested Skills: Javascript especially familiarity with the "`${}`" template syntax https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals