The Problem

When a bank takes ownership of a property, such as when it buys a property at a sheriff's sale or foreclosure auction, it takes liability for all of the responsibilities of ownership. These include paying property taxes. As such, once the bank takes title, it'll become the owner of record for the property and will be responsible for all property taxes that were past due at the time of the sheriff sale.

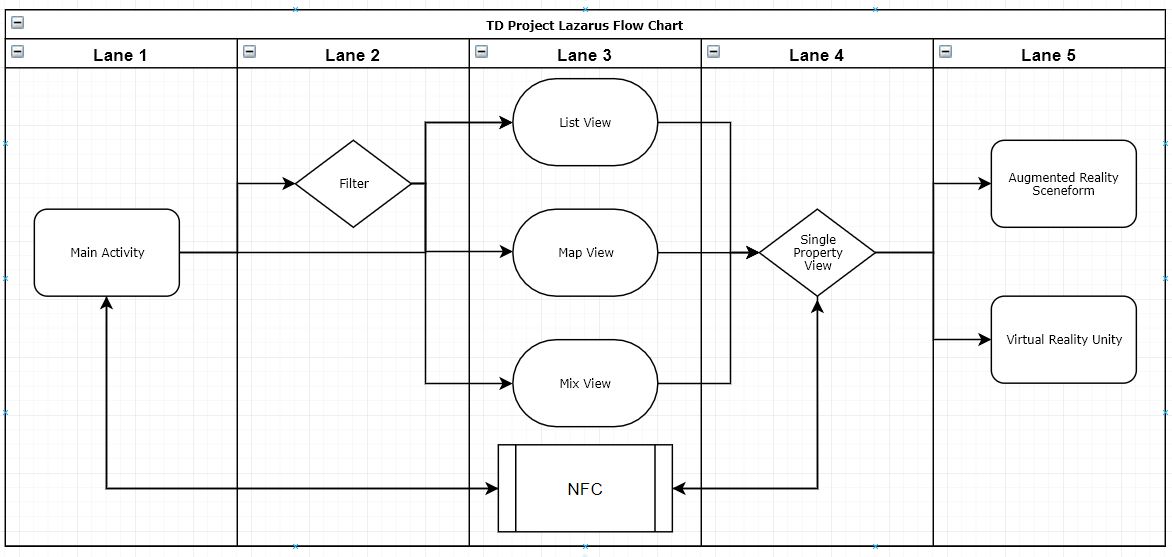
Where did you get the data?

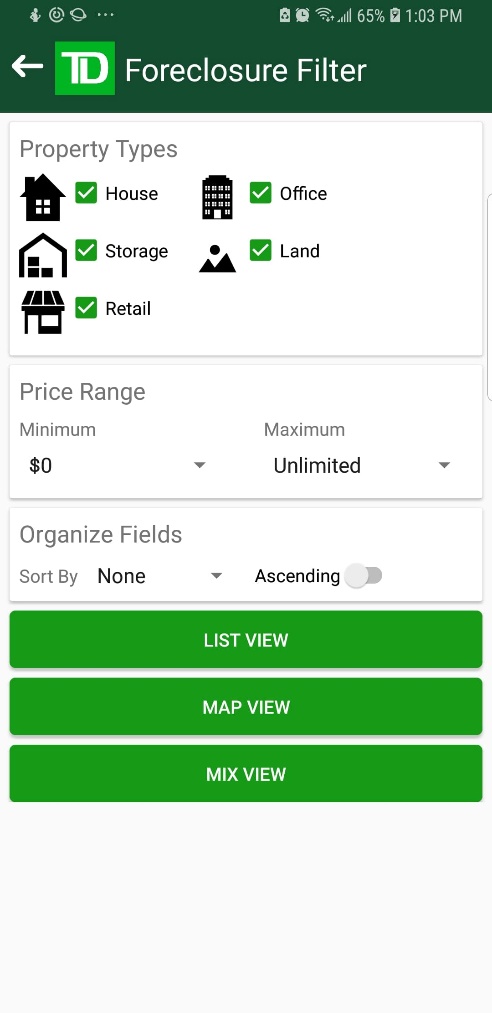
After doing a quick google search on where to find a foreclosure listing within TD. I came across a helpful TD support form where a User asks “Where do I find a list of td bank foreclosures”. They then reply with a link to a 2-page pdf document of foreclosure listings. The pdf contains a total of 117 properties there are 62 houses, 30 lands, 13 offices, 11 retails, and 1 storage. The problem with this is that the clients will have to copy and paste each address into google just to see an image of the house. It takes up a huge amount of time if you constantly looking a multiple property. Wouldn’t it be nice if there was a better way of doing this?

The Solution

The idea is to improve what is already there by having an application that will help clients view properties more easily and comfortably. This will create more investment opportunities as well fill vacant properties. It will also save a huge amount of time and hassle when trying to find the right property. As well as remove the unwanted liabilities of paying property taxes for the banks. The application will help users find exactly what they are looking for with the filtering system. Also, instead of driving all the way to the property. They have the option to view the property layout in augmented reality as well as virtual reality.

Implementation

All views and the filter fragment are stored within the main activity’s fragment manager. The reason for this is to have a drawer navigation panel that will consist throughout the program.

**Filter View**

Main features

* Select the **Property types** you wish to view
* **Price ranges** are apart of a spinner because it is easier and faster for user to select a drop down than it would be to type the full price range in for both fields.
* **Organize fields** are none, distance, price, state, and size

**Price sort** field is simply checking if the values are within

the same range

**State sort** field is sorting the by alphabetically. Also, their

will be another spinner that will be spawned

programmatically. That will grab that data from firebase

but the values that are stored are the short versions of the

state. For an example Florida is stored as FL. So, there is a

look up table to grab the long version.

**Size sort** field is sorted by property size but some states are

In sqft and acres also all the values are in strings. So you

have to grab all the values before until you hit a character.

Then you need to figure out if the value is in sqft or acres.

After you have the number and the size type you need to

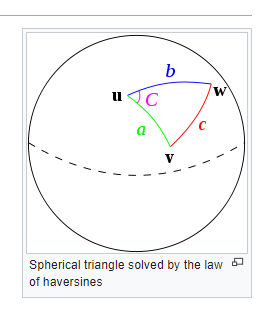
Convert them into the smallest possible measurement

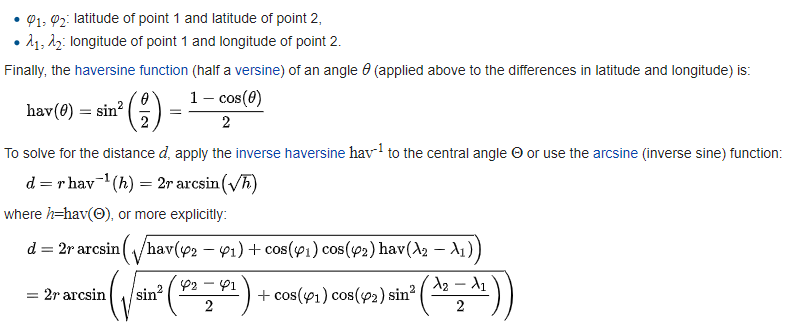
which is sqft so if you do have a value of acres you will

need to times the value by 43560 then compare the values

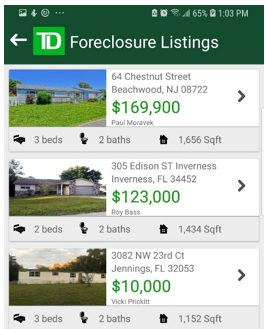
**Distance sort** field is calculated with the haversine formula

<https://en.wikipedia.org/wiki/Haversine_formula>

The haversine formula determines the great-circle distance between two points on a sphere given their longitudes and latitudes.



**Ascending/Descending** switch just swaps the sorted items index from forwards to backwards

** List View**

The list view is a fragment that contains a loading screen with

a spinning gif with loading text. It will be replaced as soon as all

data that has passed the filter system has been set. It will be

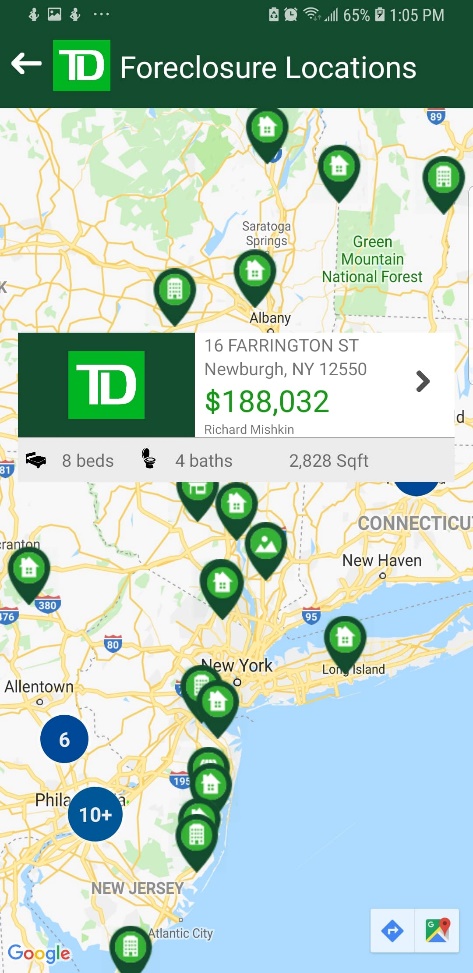
swapped with a **Recycle View.** Therecycle view is kind of like

list items but the neat thing with this instead of swapping text

you can swap in a custom layout. When the data is pulled it is

inserted into a custom-made list item view and dynamically

added.



**Map View**

After values are pulled and filtered. The data is then drawn to the

map. Each map **icon** is **custom made**.



User Homes Offices



Market Storage Land

Each icon when pressed shows a custom information window that is

dynamically rendered. There is an image in the **custom information**

**window**. Also, all icons are **cluster markers**. So, when you group

many markers together instead of it getting overwhelming you can

have a number displaying the amount that is there. You can press

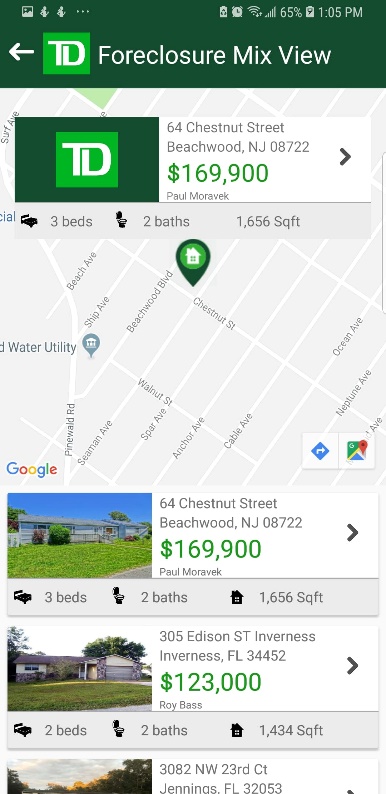
on the cluster and it will animate the camera to view a closer up

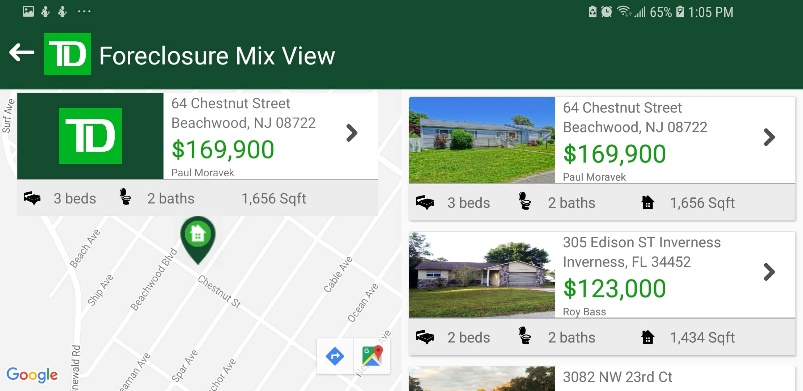
view of the cluster.

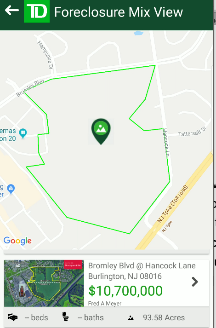
Also, **Geolocation** was used to grab the locations from the address.

But it would take **3 seconds** to load all the values. It wasn’t fast

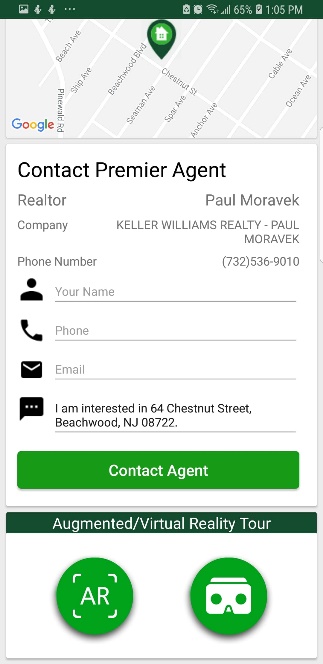
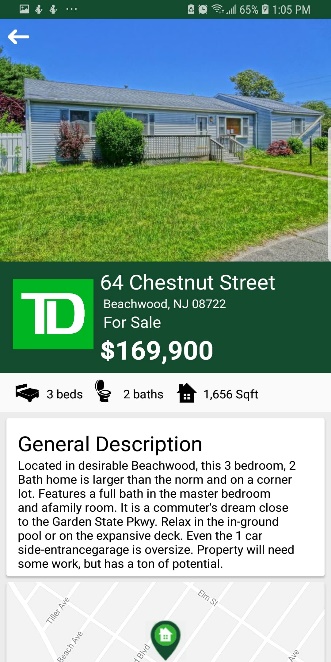
enough so we decided to place them inside of the firebase database. The way we did this is create a log that would capture all the addresses and put them into console from this list we made another program to take in two text files one for the location and one from the firebase data exported as json. The program would add the values to the json file and just re exported back to firebase. The app will grab the **data and display immediately**.

**Mix View**

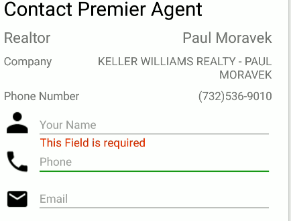


This view is **mixing both** the **list** view and the **map** view **together** so you can select an item in the list view and it will animate to the location. You need to find where the index is of the current property. But it is sorted in the list also you have to figure out where on the map is this property then animate to that location. This view has **dynamic constraints** when the user goes from landscape to portrait. When you view Bromley blvd @ Hancock lane the land boundaries are drawn accordingly.

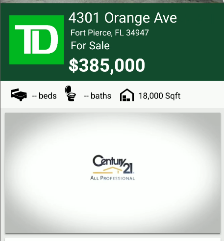
Property Information

This window is an activity that will display the data of a selected property. There is a bunch of things going on in the background. Such as gesture, NFC scanning, text field validation, Media player, as well as a video play if you select a storage property.

- **general description** of the property

- **Text Validation** is dynamic and will spawn if the user doesn’t type a value into the field or is invalid.

- If you press the phone icon you can **call** **the number immediately**

- If all fields are typed in you can start an intent to send an **email**

- If you press the sms icon you can start an intent to **send an sms**

- If you are looking at a storage property you can have a **video play** for the property.

- **Gesture** is a flick from left to right to go back. Kind of like **tinder** when you like.

-Two buttons for **AR/VR at the bottom** if you are on the property 64 Chestnut Street

**Augmented Reality view**

This activity is using **ARCore** with **sceneform** to render a 3D model of the property you are currently looking at. Also, if your device isn’t compatible you will get a notification and redirected back to your last view.



**Virtual Reality View**

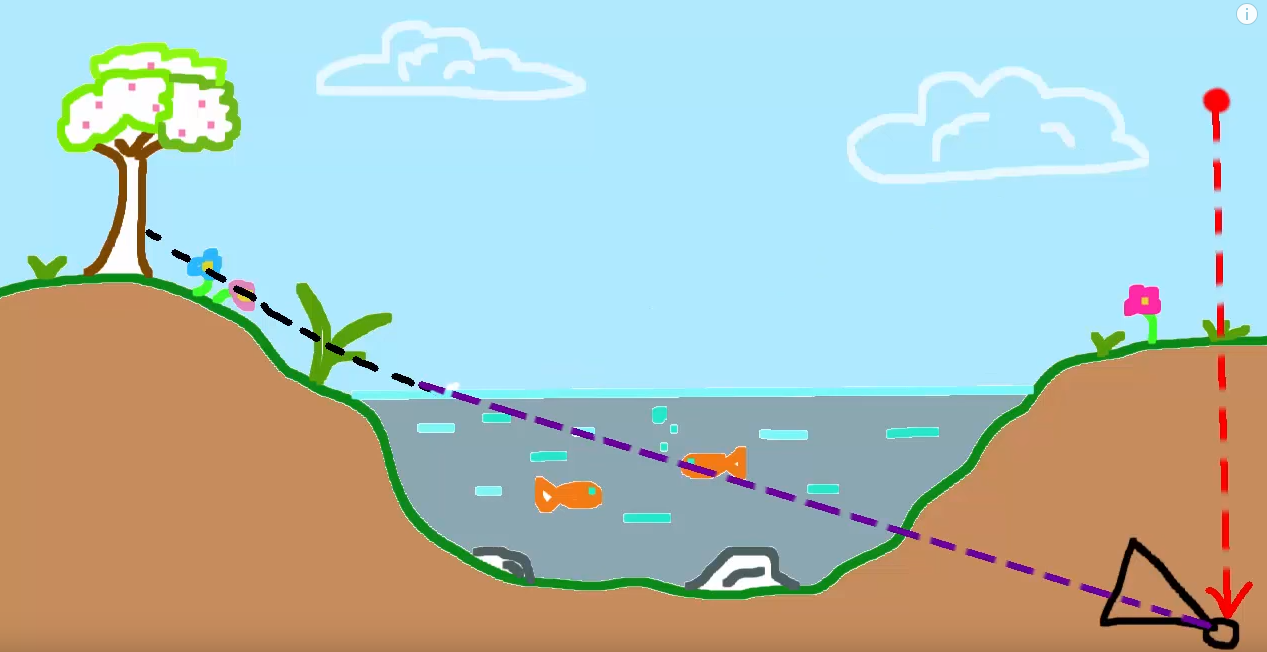
This is an application inside of another application this uses the **Unity** engine to walk around the environment. This uses **google cardboard** with a **Bluetooth controller**. **But** for the **live demo** I will be using **mobile input** controls just so we can share the screen to the projector.



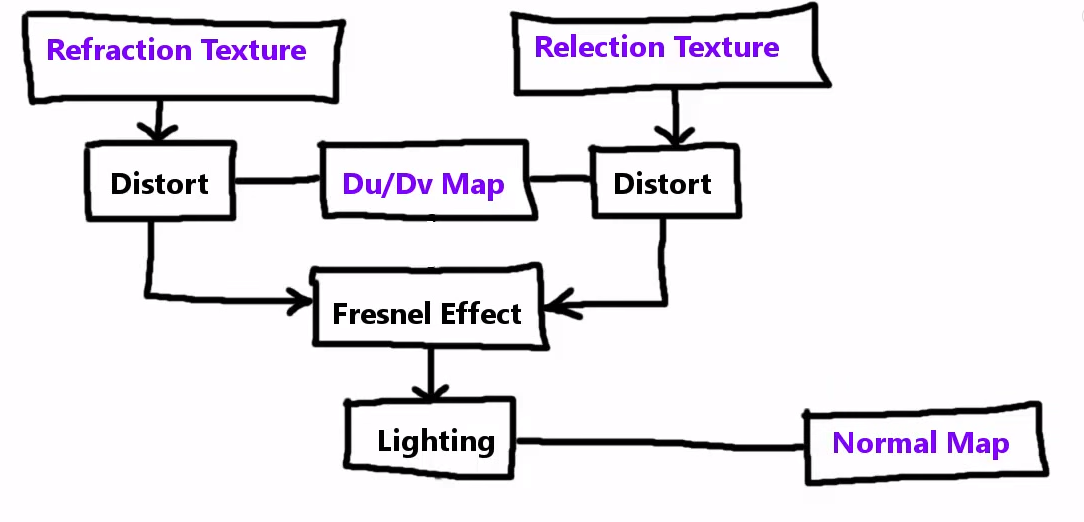
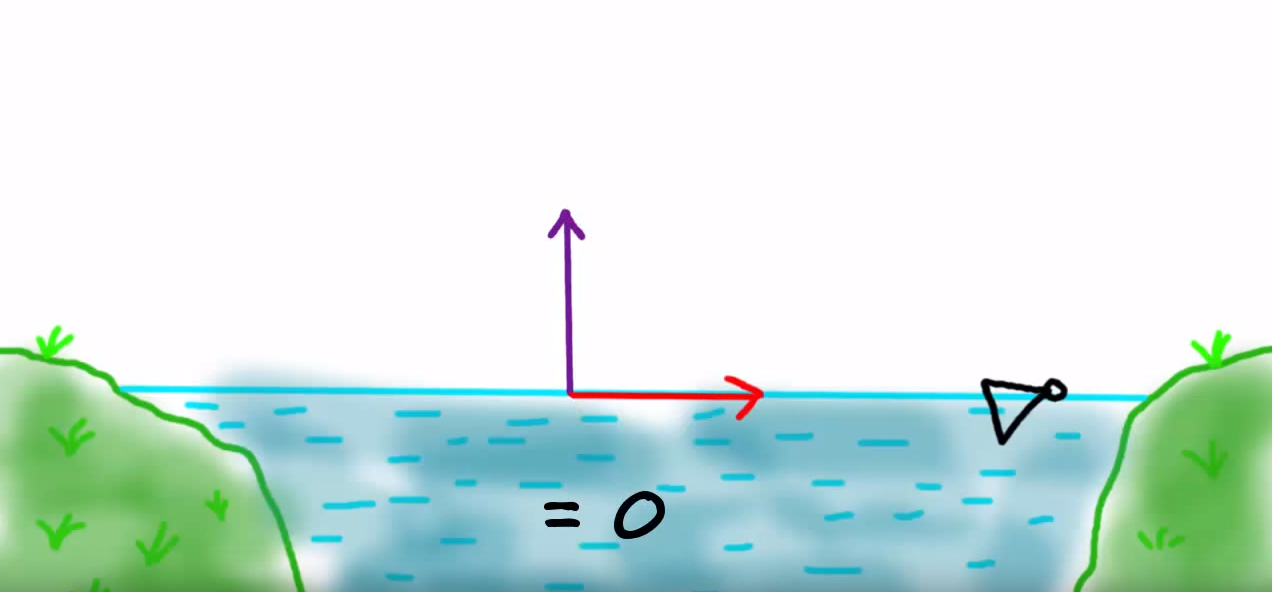
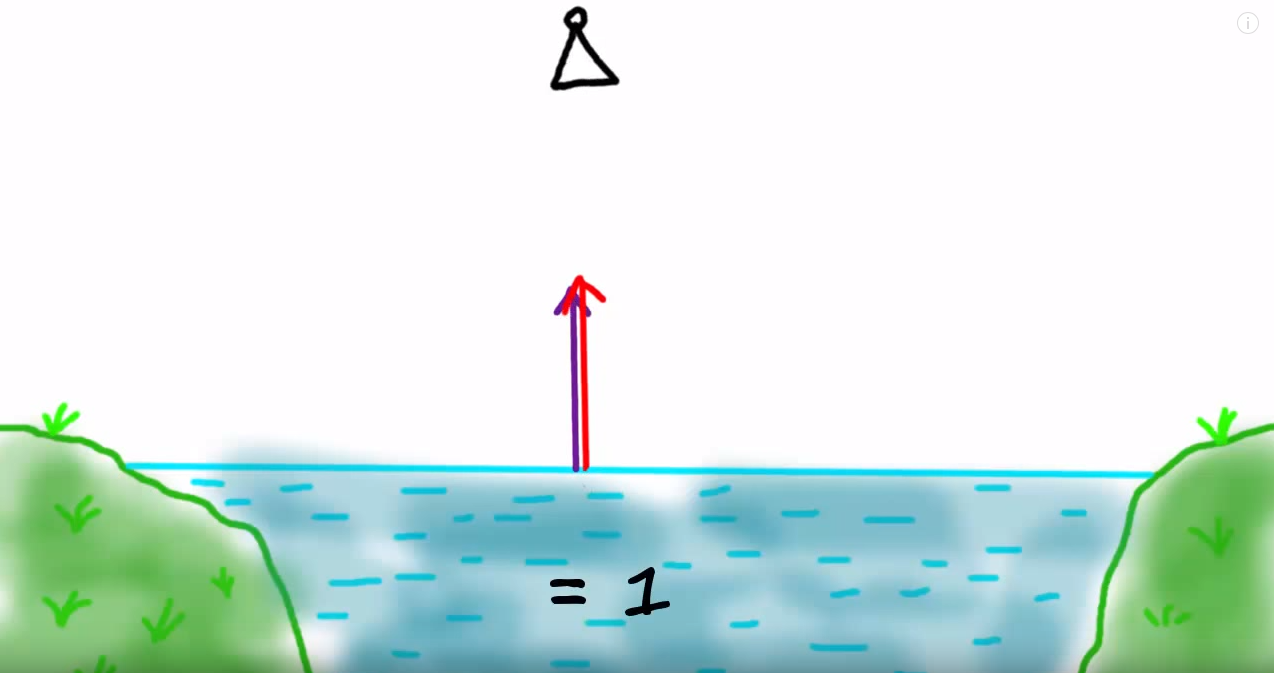
**Unity Behind the Scenes**

**Water refraction**

**Reflection** is achieved by first figuring out your position to the water plane. After you find this out. You then need another camera under you to figure out what is to be rendered on top of the water layer. The camera that you are using to render what is below the water level.



We then trim everything below the water level in the second camera to decrease the amount of resources used. Then we render what is left on top of the **water plane**.

Depending on where you are looking at the water depends on the amount of refraction you are getting from this. For example, when looking at water straight down it will look like the top level is completely transparent. But when you view the water on a lower angle you can see the reflection kind of like a mirror.

Another thing is when your camera goes below a certain height you need to add noise have partial effects to the screen of the camera to simulate being under water. **Add noise** to the camera as well as **bubbles**. After when going outside the pool there is a **2D texture** that slowly loses transparency. As well as another sprite for lines of water going down the camera.

Plus, there is a **lens flare** when you look at the sun depending on your current location to the sun it will display **sun beams**.

**Minor Details**

**Buttons**

The buttons are custom made to match the exact border-radius of 4px that are used from the easy web layout. Also, the Augment and Virtual reality buttons are custom made from photoshop.

**Scalable**

The app can run on any android phone device with the constraint layout

**Rotation**

The app supports portrait as well as landscape

**Transitions**

All animations are custom made slide when going forwards and back between fragments.

**Shared Preferences**

Sort fields that have been selected are saved into the shared preferences so if the user closes the application, they can keep their old sort filters. If nothing is found the user is notified immediately.

**Drawer Navigation**

Is consistent when moving between fragments also past fragments are stored into a Fragment Transaction Back Stack.

**States Names**

The values that are stored in the firebase database are only short forms so there is a look up table to get the long version of the names

**Splash Screen**

The splash screen is the same as the one used in the TD My Spend app.

**Loading screen**

A custom loading screen when loading the list view items.

**Firebase**

All the data is stored into a firebase data base

**NFC**

NFC is used if you wish to save a property as a bookmark you go view it in the property information and scan the card to save it. Or you can view the saved property on the fly within any view.

<https://console.firebase.google.com/project/td-project-lazarus/database/td-project-lazarus/data/>

Libraries used

**Glide** – loading images from url

**Firebase** – a database mobile platform that helps you quickly develop high-quality apps

**Google maps** – with clusters to show the properties

**Gson** - Json but google version so that you have a way to send data as a string package

**Sceneform** – for ARCore

**Unity** – for Virtual Reality Tour