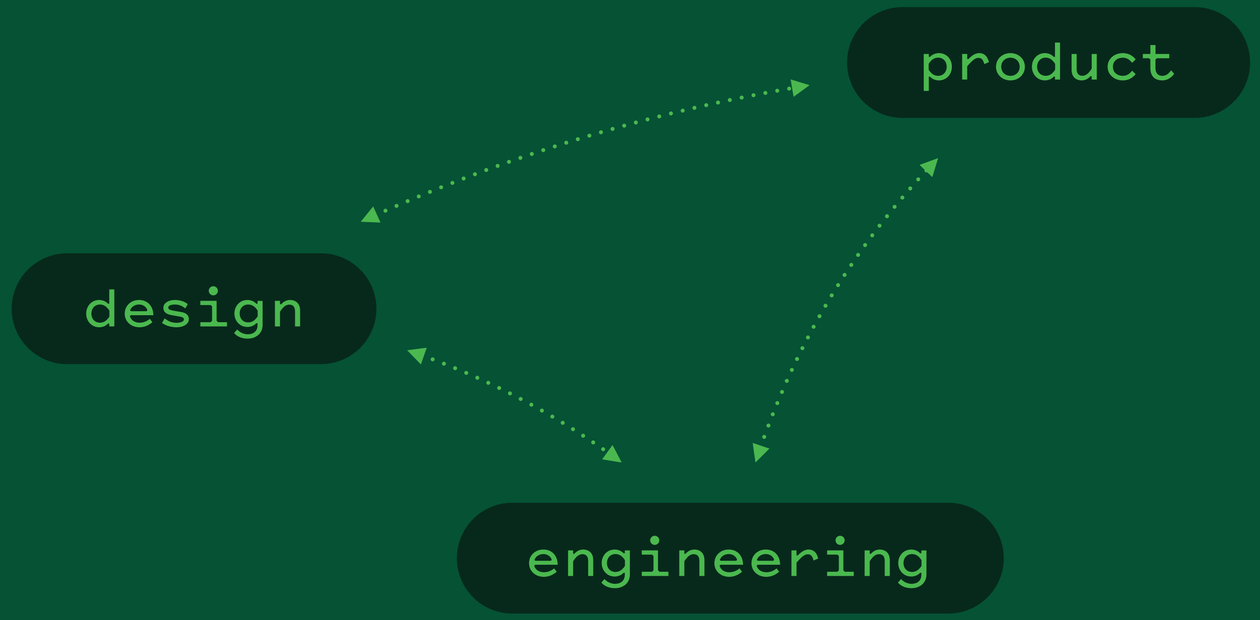
Basheer Tome

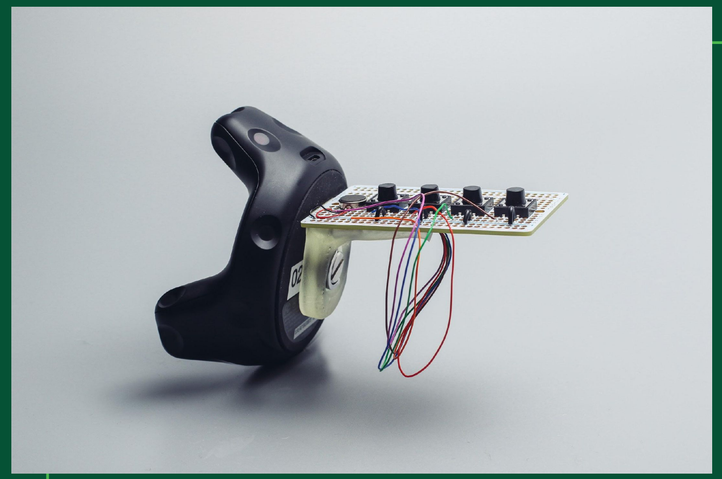
Portfolio 2020

I'm a hardware interface designer working from end to end to invent new hardware products. I shepherd them from sketch to breadboard to pixels to launch.

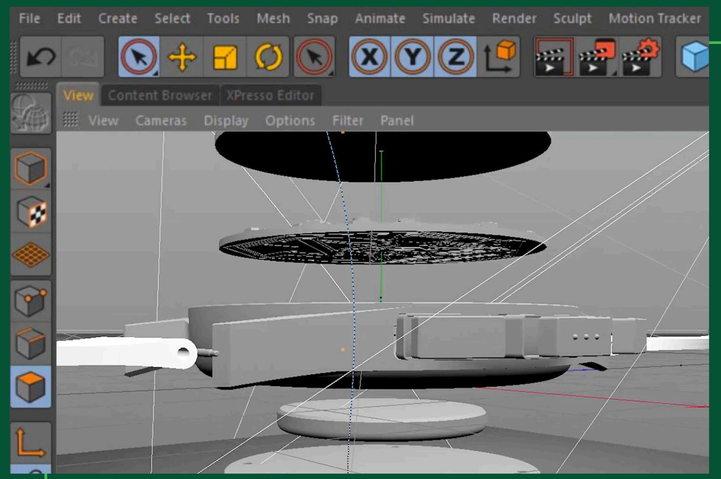
I serve as a strong bridge across a wide array of functions to infuse crisp product intent, usability, & beauty throughout the product holistically.



To do that, I use a blend of:



electronics hacking



filmmaking tools

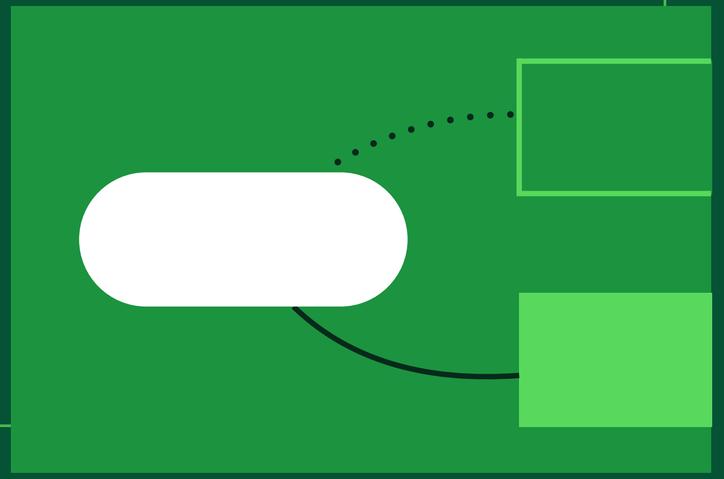
physical prototyping

Combined with:

```
//set single LED
app.use(‘single/
app.get(‘channel
console.log(setL
```

scrappy programming skills

ui design & wireframing



a love of flow charts

My expertise has been built upon an academic foundation in HCI + ID:

2009
2013

Georgia Tech

B.S. in Industrial Design

Learned the fundamentals of form-giving & problem-solving with design while spending my summers interning in UX in the Bay Area at Google.

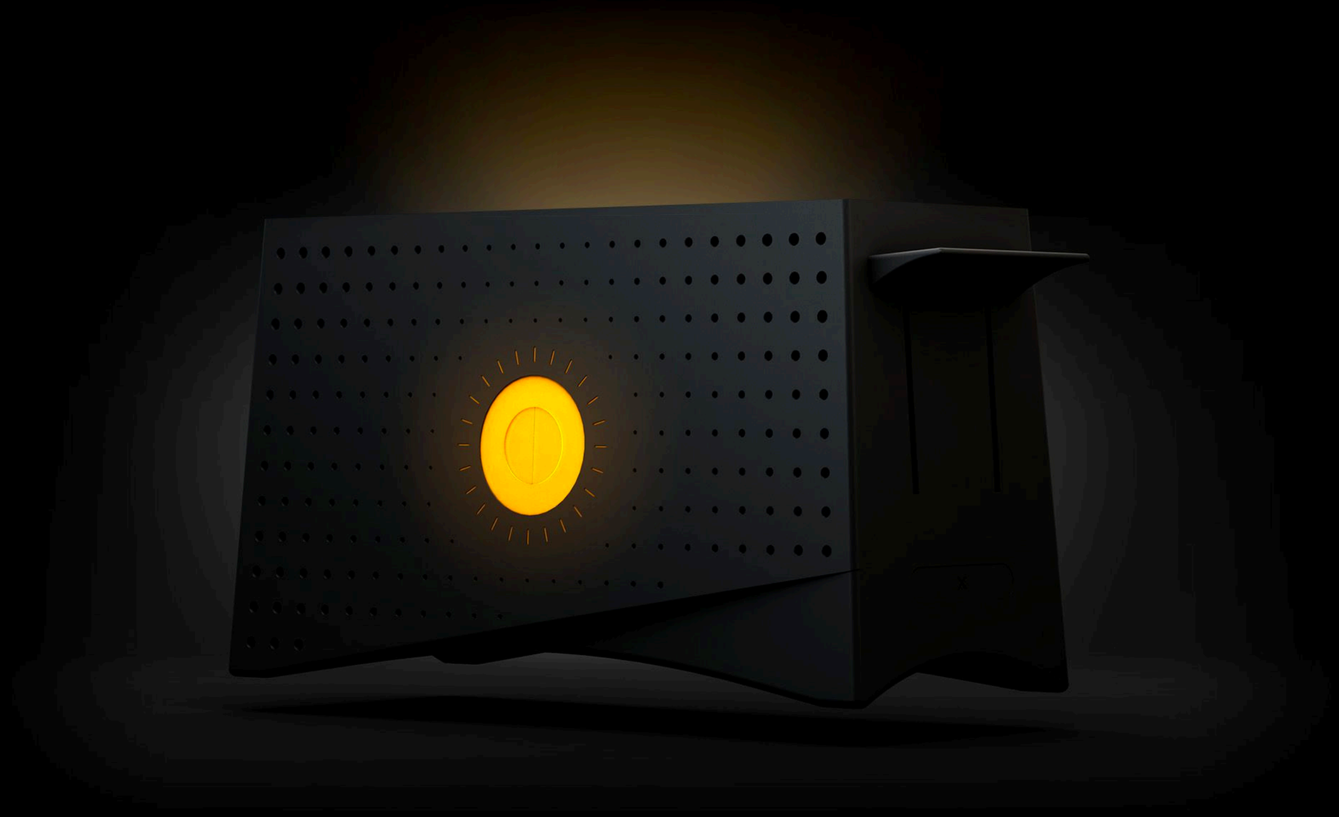
2013
2015

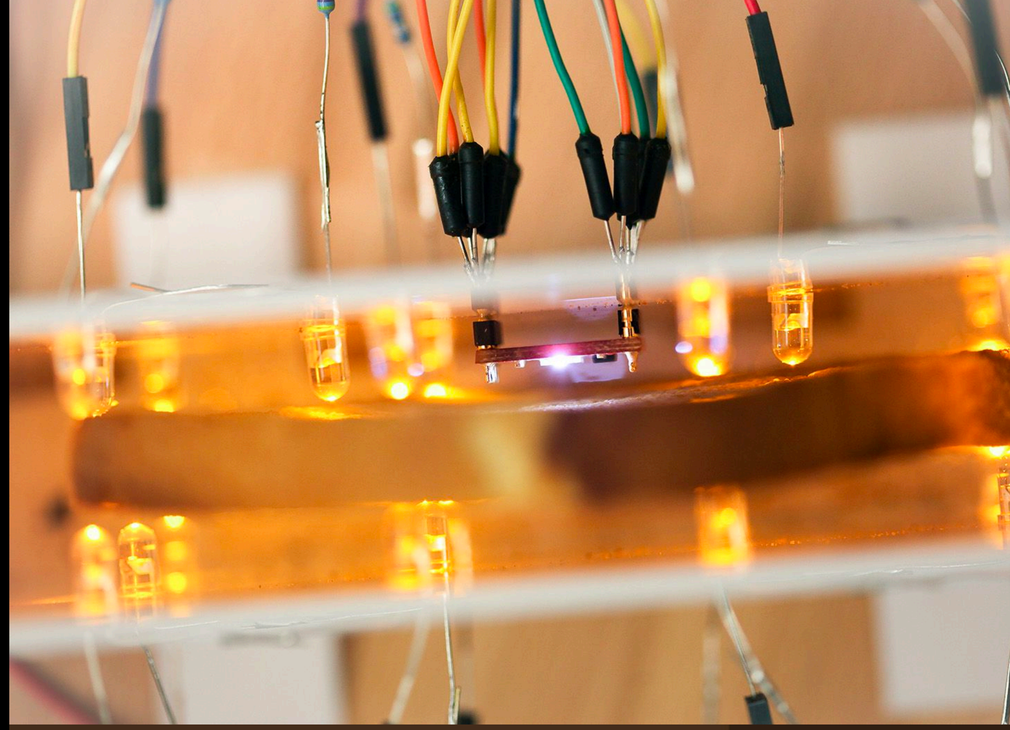
MIT Media Lab

M.S. in Media Arts and Sciences

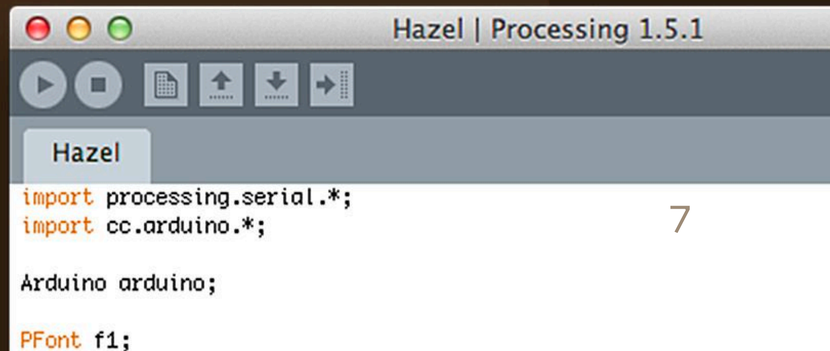
Studied in the Tangible Media Group designing new automotive interfaces and programmable materials for companies like Jaguar, Lexus, and New Balance.

Georgia Tech

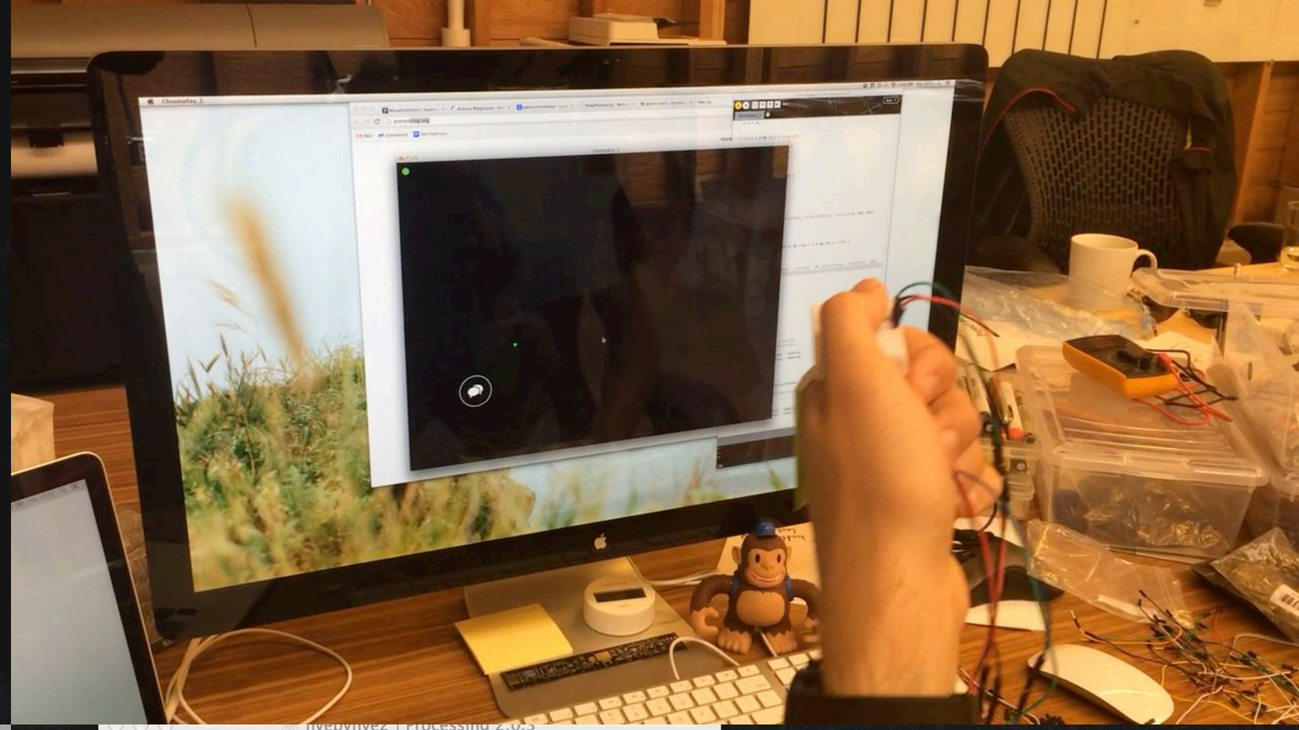


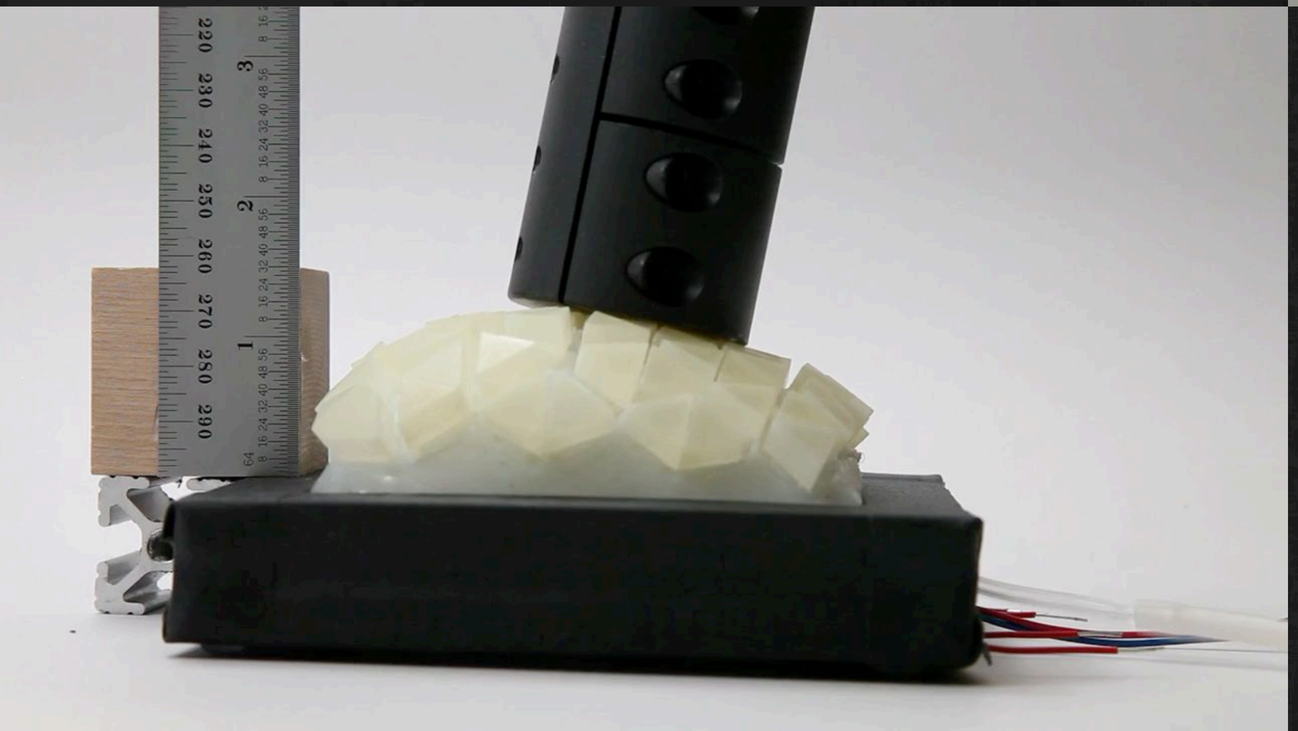


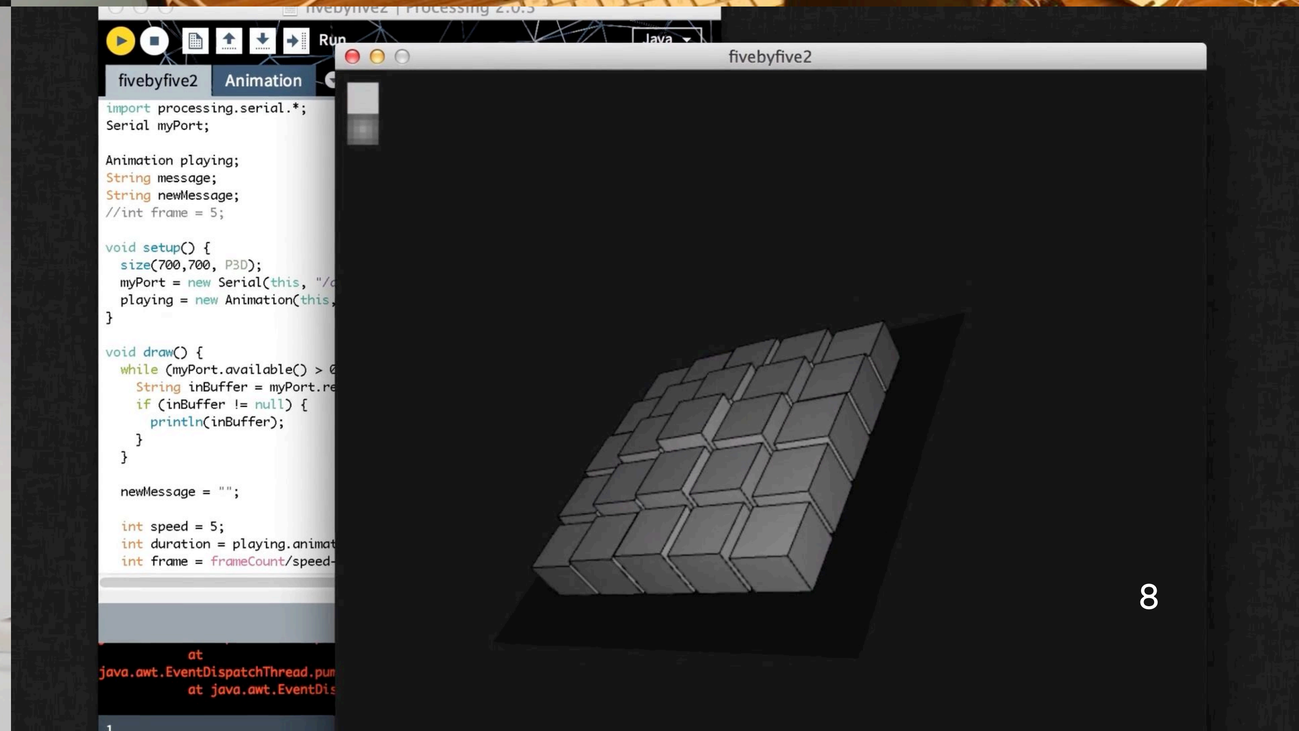
74



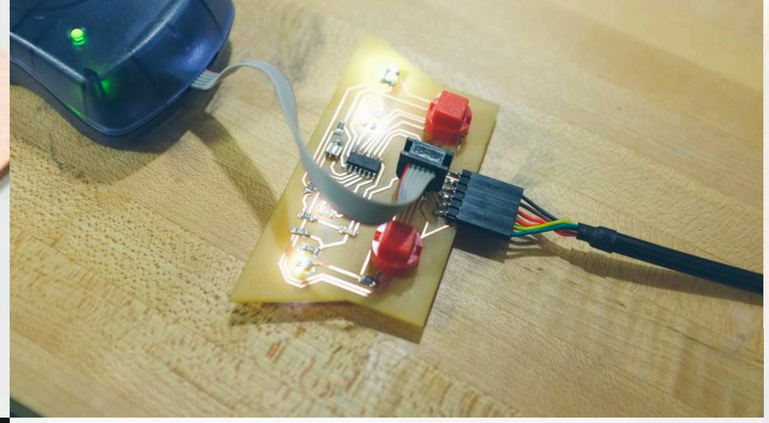
Media Lab

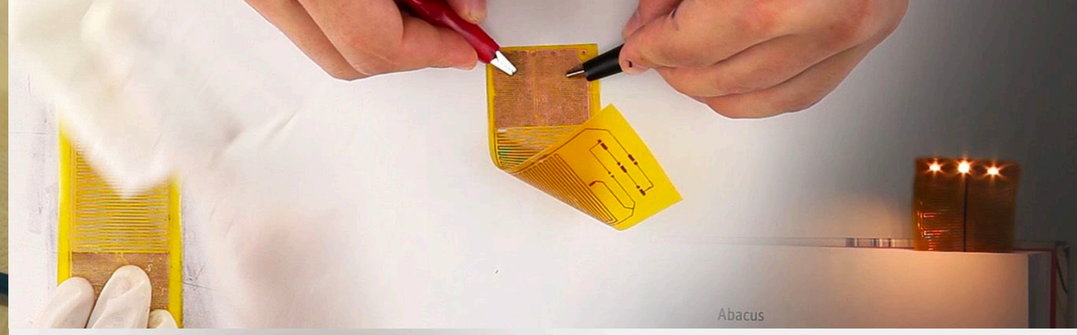


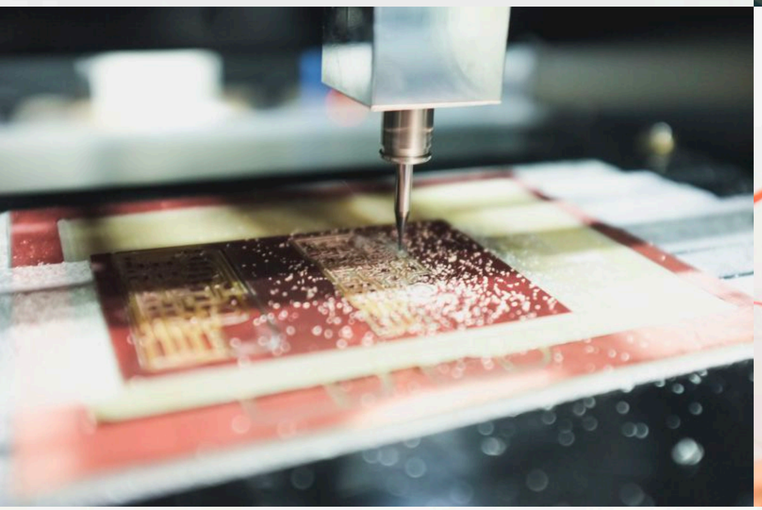


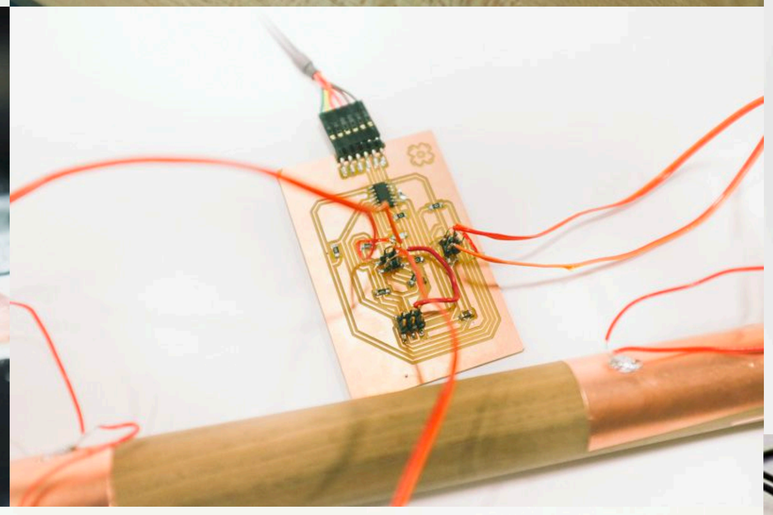

fivebyfive2 | Processing 2.0.3
Run
Java
fivebyfive2
Animation
import processing.serial.*;
Serial myPort;
Animation playing;
String message;
String newMessage;
//int frame = 5;
void setup() {
size(700,700, P3D);
myPort = new Serial(this, "/
playing = new Animation(this,
}
void draw() {
while (myPort.available() > 0
String inBuffer = myPort.re
if (inBuffer != null) {
println(inBuffer);
}
}
newMessage = "";
int speed = 5;
int duration = playing.animat
int frame = frameCount/speed-
at
java.awt.EventDispatchThread.pum
at java.awt.EventDis
fivebyfive2

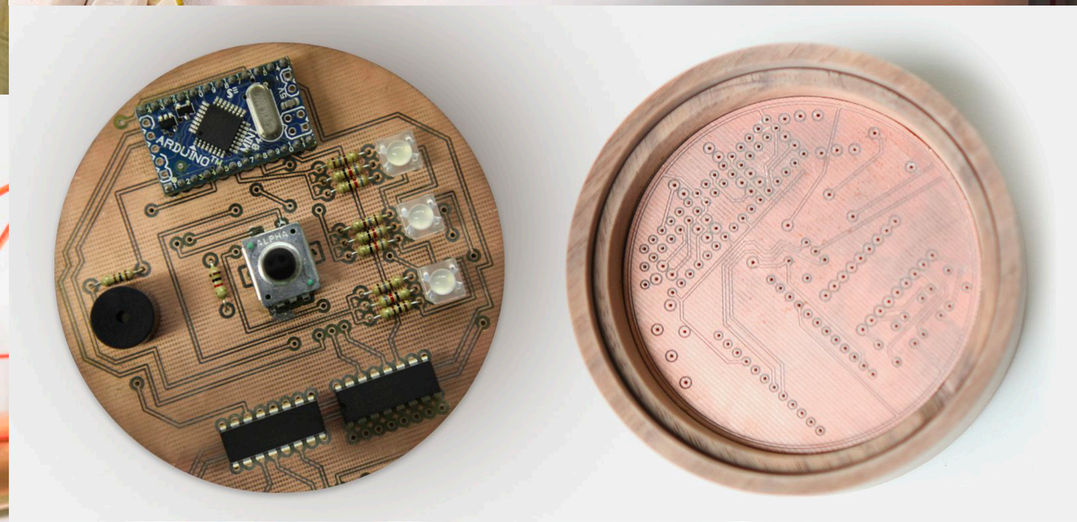
Media Lab

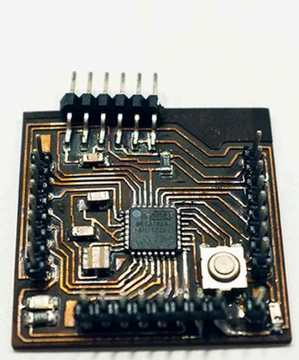


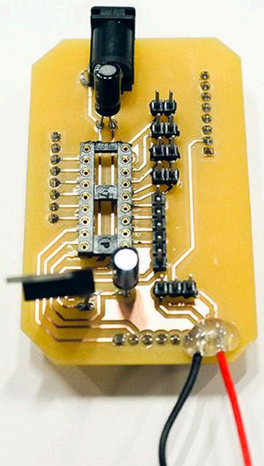


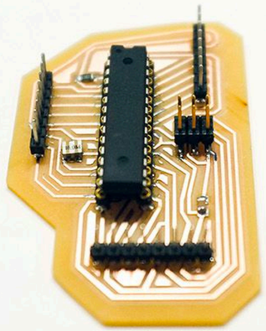


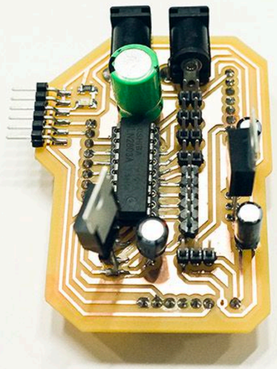


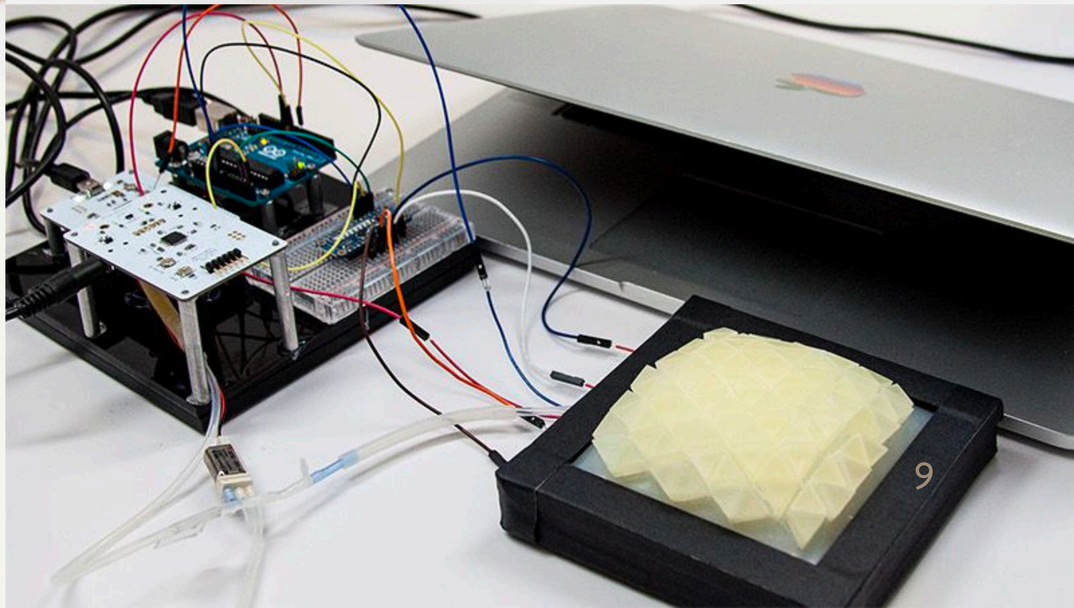












Clutch Knob

The Clutch Knob senses gaze and adapts both a visual & physical interface to match the acuity of the user depending on their direct or indirect attention.

video

I then put those skills into practice and refined them during my time at Google.

After hours...

you'll find me pointing my camera directly into light sources, blow torching food, or playing with and writing about new kitchen tools.

Case Study

Daydream Controller

Google - 2016 + 2017

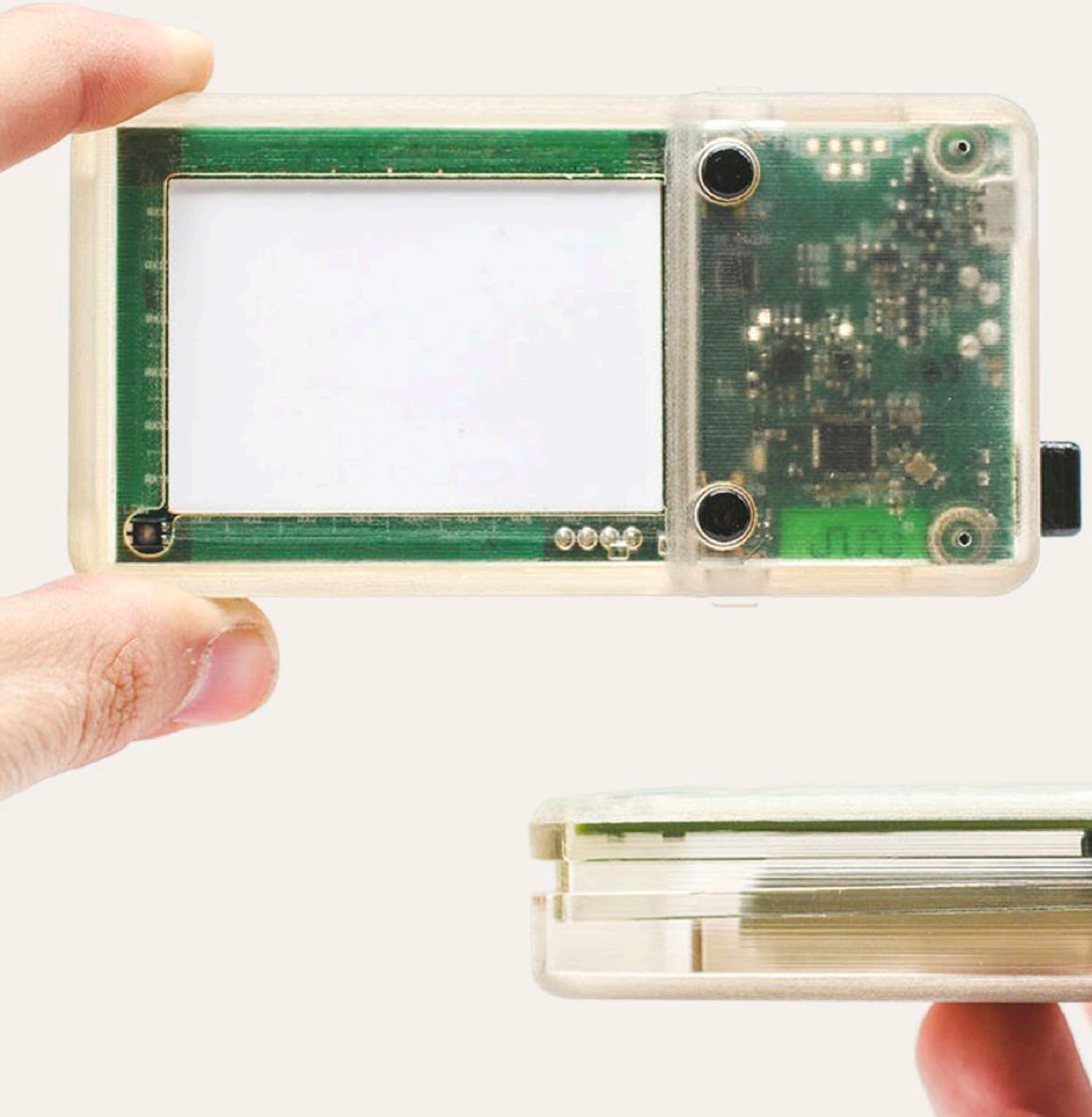
Daydream

Aimed as a premium upgrade to the Cardboard product, the Daydream View sought to solve the two biggest pain points: **comfort** and **control**.

As the design lead for the controller, I balanced working simultaneously on the industrial design, hardware experience, & software experience.

Daydream

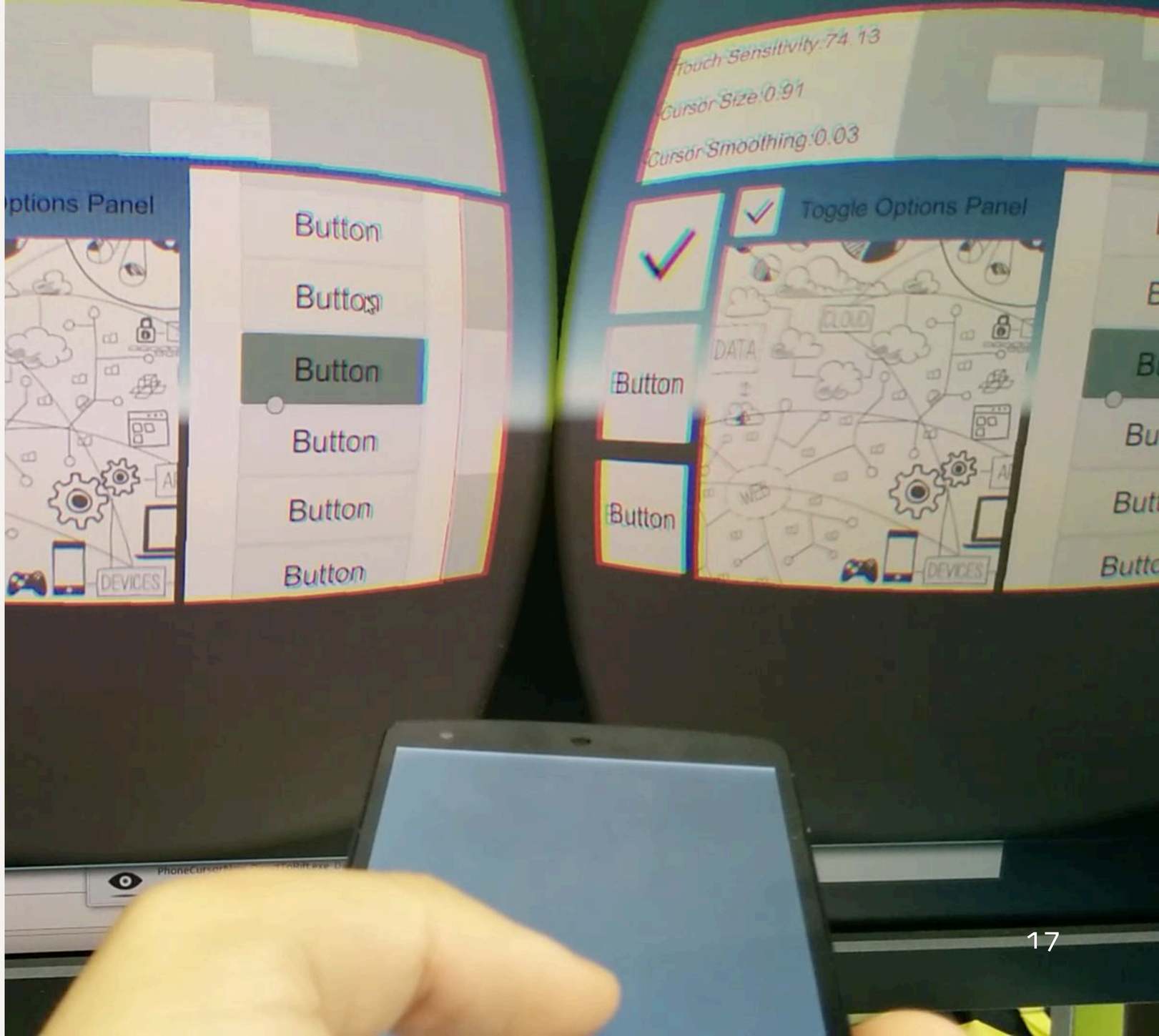
Once we had our initial concept white boarded, I started building a barebones interaction model with simple hobby parts.



I then collaborated with electrical engineers to iterate into a more formalized & robust prototype that we could give to our developers.

Daydream

At the same time, I worked directly with the software engineers in Unity to prototype the pointing interactions using a phone and to build a simulator that our extended network of international of games developers could download & start using themselves.



Given the differences in form factor, I laser-cut & shipped physical overlays to prevent those external developers from accidentally optimizing their touch gestures for the full phone’s screen size rather than the much smaller and circular touchpad in the real final product.

Daydream

In parallel, I used a deep series of sketches, renders, rapid physical prototyping, and clickable models to get the industrial design off the ground.

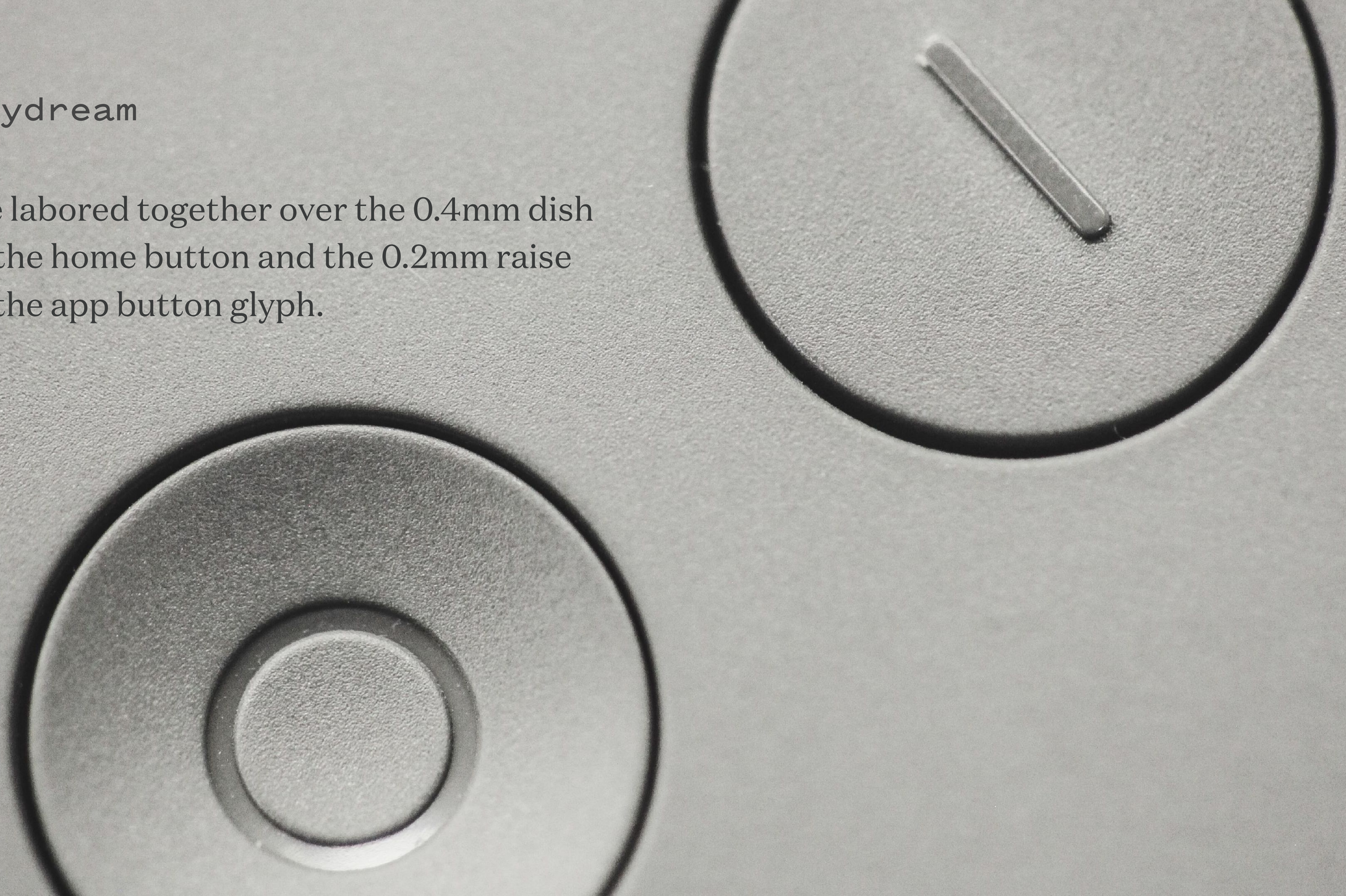
The clickable & weighted prototypes enabled the teams to plan component locations and a ballast to push the weight further back to optimize comfort.

Daydream

I then moved onto detailed explorations to design & test the shape & feel of each of the buttons. People would be blindfolded while using the controller so it was critical each interaction point have a clear tactile feel.

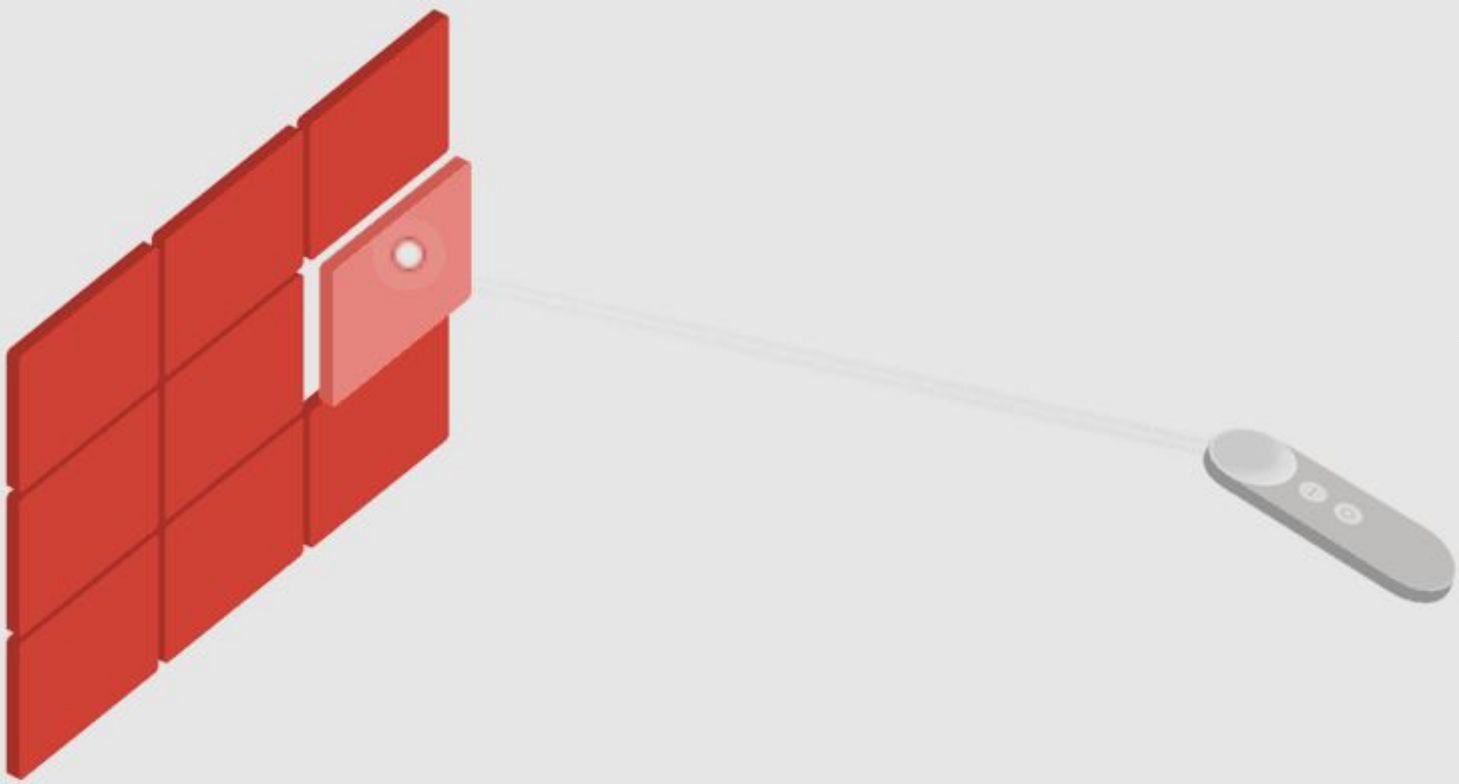
Daydream

We labored together over the 0.4mm dish of the home button and the 0.2mm raise of the app button glyph.

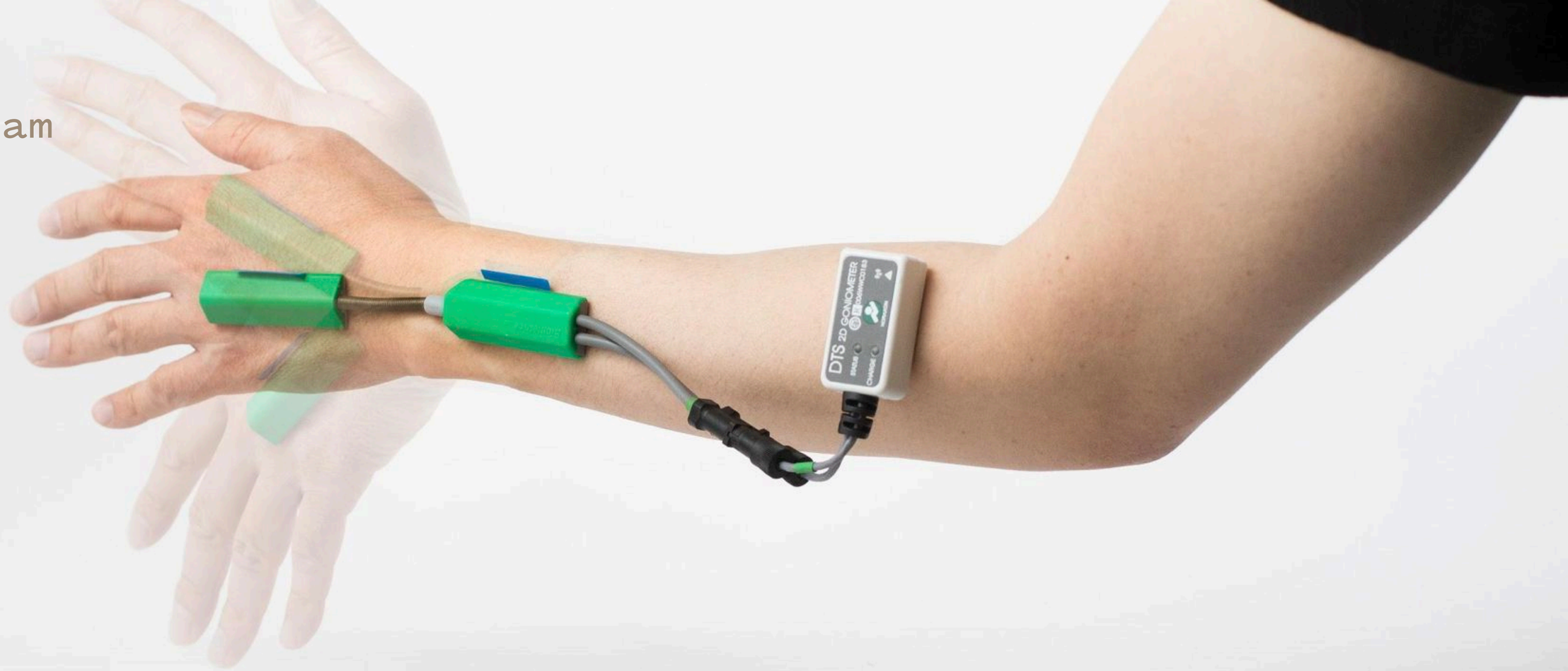


Daydream

And validated the designs through a series of machined prototypes to balance comfort & having enough tactility.



Once we were near launch and deep into our tests, we realized there was a discrepancy between where people were physically pointing and what they saw in VR.



So, I partnered with our human factors researcher to build prototypes that we could measure & test wrist angle with. We found that the natural pose of the controller was 15º higher than parallel. Once we adjusted that in the UI, everything felt good again.

My external talks that give even more depth:

Physical Design →

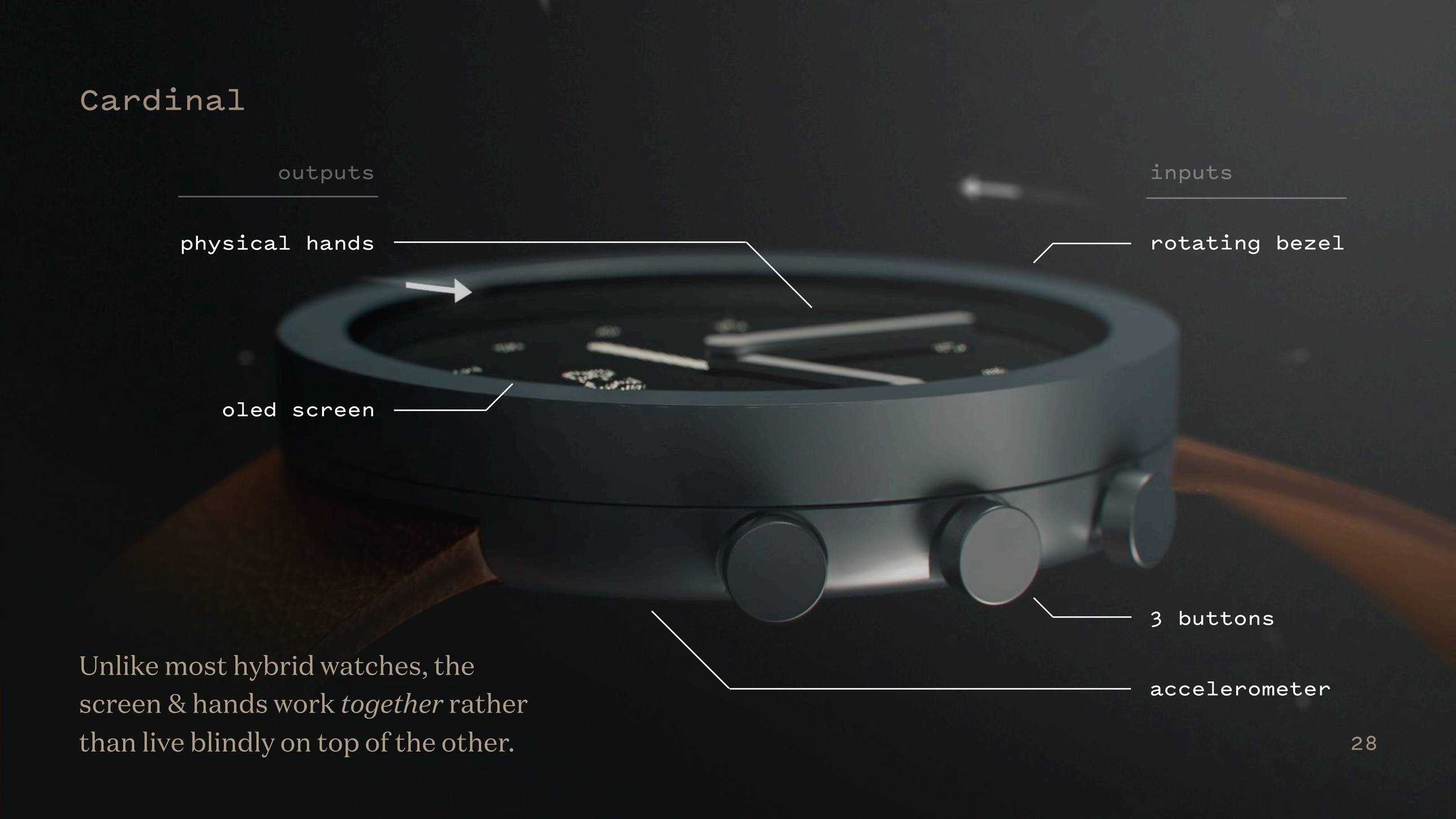
Interaction Design →

Case Study

Cardinal Watch

Personal Project

Cardinal



Unlike most hybrid watches, the screen & hands work *together* rather than live blindly on top of the other.

Ambient & Contextual

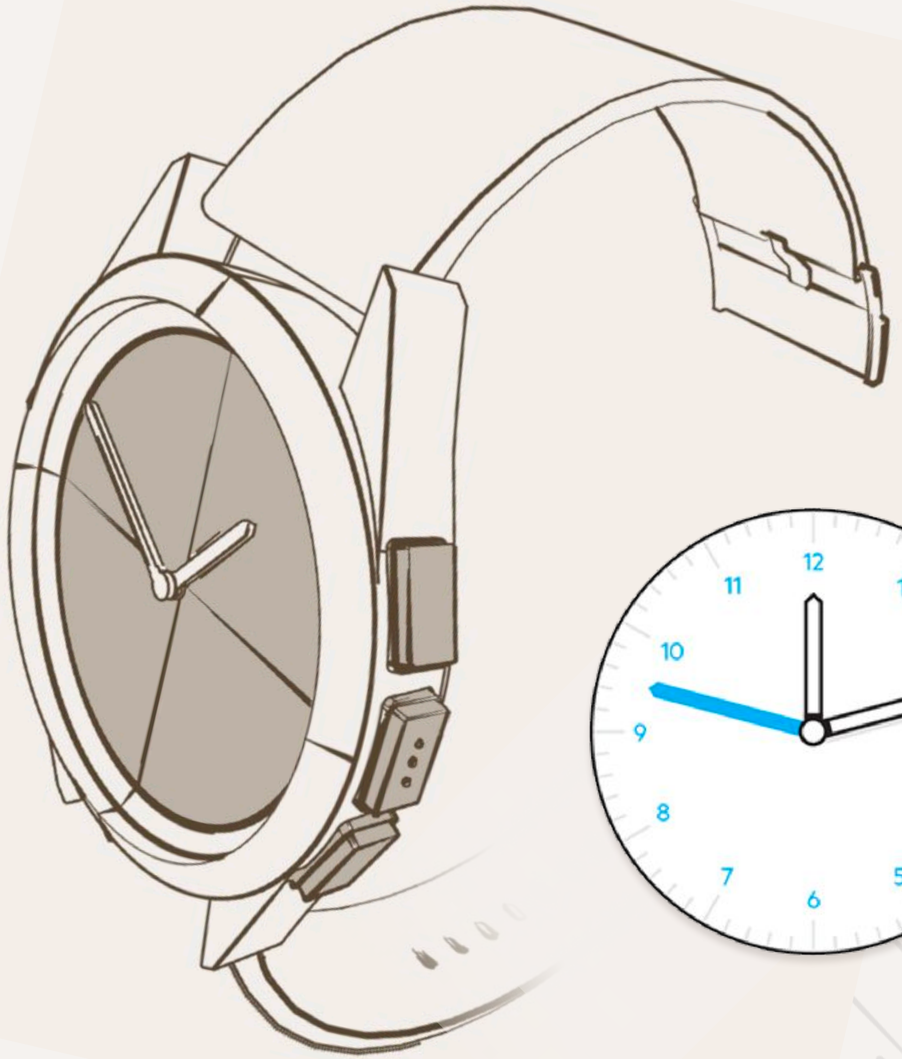
Cardinal focuses on 3 key parts of your day: when, what, & where

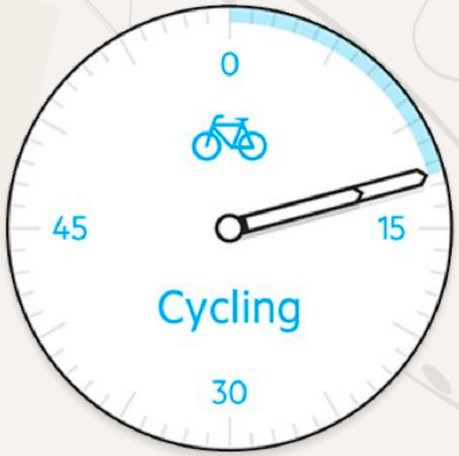
Both the display and the hands change based on your current activity. When you’re walking or running, the display automatically shows how long you've been doing that activity.

Flick your wrist to show time again.

Process

Through a series of hand sketches, ia charts, and wireframes, I mapped the key user need areas (**when**, **what**, & **where**) into three main modes, each of which with its own dedicated physical button.





Cardinal

time with timer set or stopwatch running

time while traveling in foreign timezome

rotate the bezel to apply a reversible time offset

city driving view in middle of the street

sitting

sleeping

rotate bezel to set new alarm time

rotate bezel to adjust and calibrate north

map view in more rural area while walking

public transit view while riding the bus, light rail, or train.

double tap the glass to zoom in to the map

tap the glass to switch to cumulative view

snowing

timer with elapsed time

rotate bezel to set timer

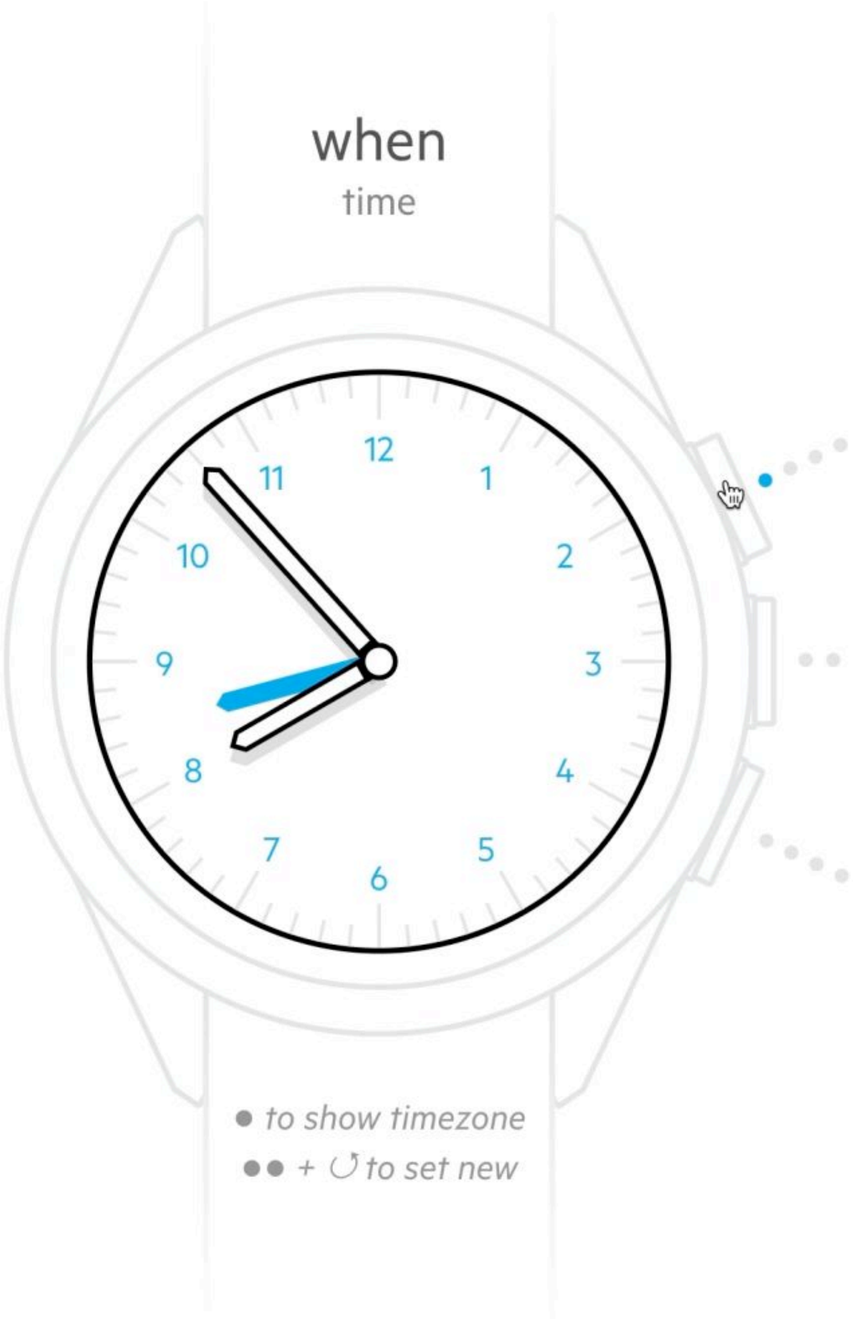
tap the glass to view other date info

I then went deeper into wireframing each mode to flesh out how alarms, dates, & timers work within **when**, how individual activities & cumulative views work within **what**, and how maps, transit, weather, and compass work within **where**.

In parallel to that more pure ux work, I also sketched, modeled, rendered, & prototyped on the physical design to better understand the physical interactions I was relying on and how that would influence the form & feel.

Cardinal

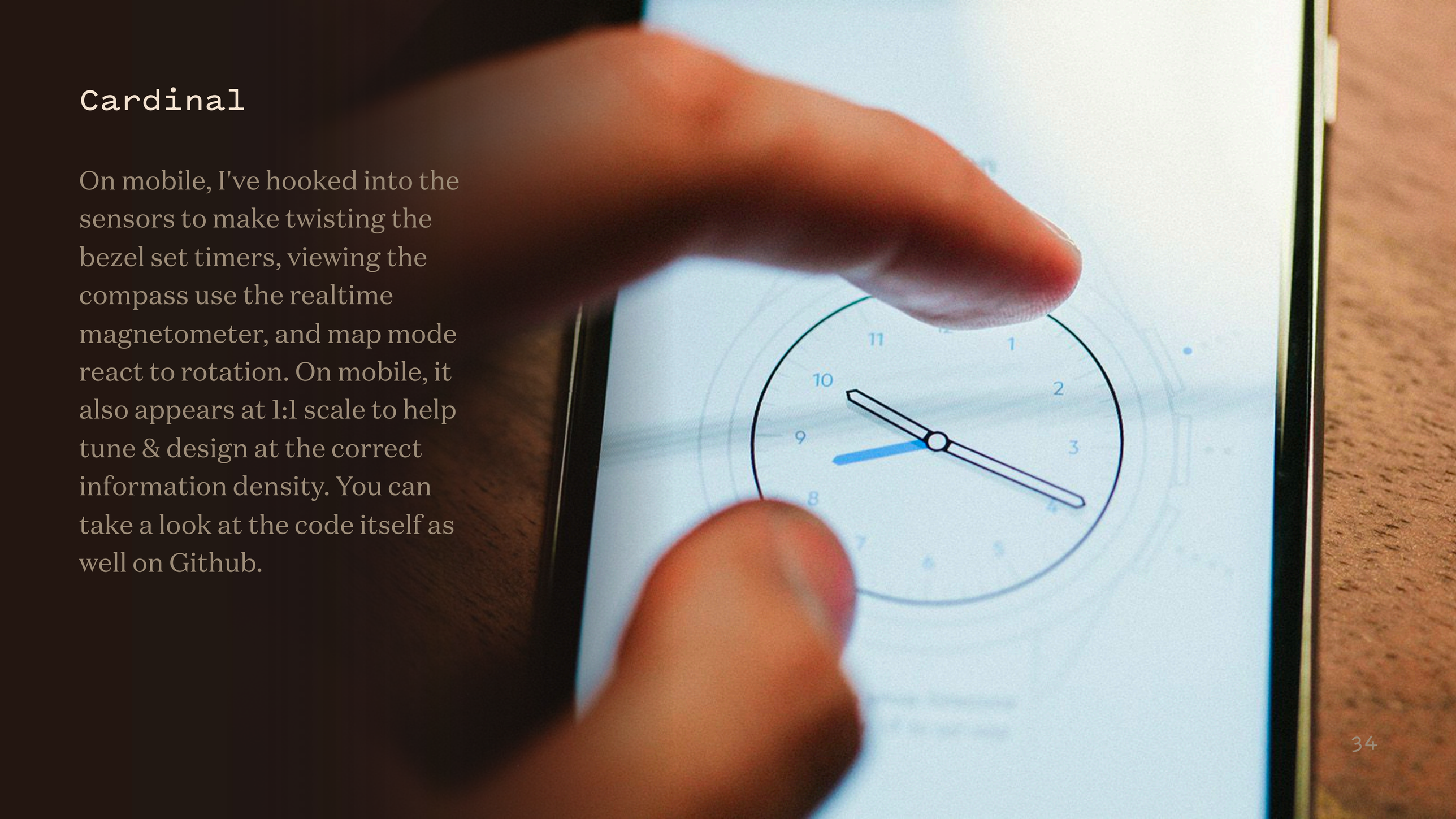
To quickly explore and iterate on the interactions, I built an interactive prototype. It's all done in javascript & canvas on the web to keep things simple, fast, platform-agnostic, and easy to share with others. While you can play with it above, it works best when viewed on a mobile phone so head over to the Cardinal test page.



cardinal.basheer.co/v2

Cardinal

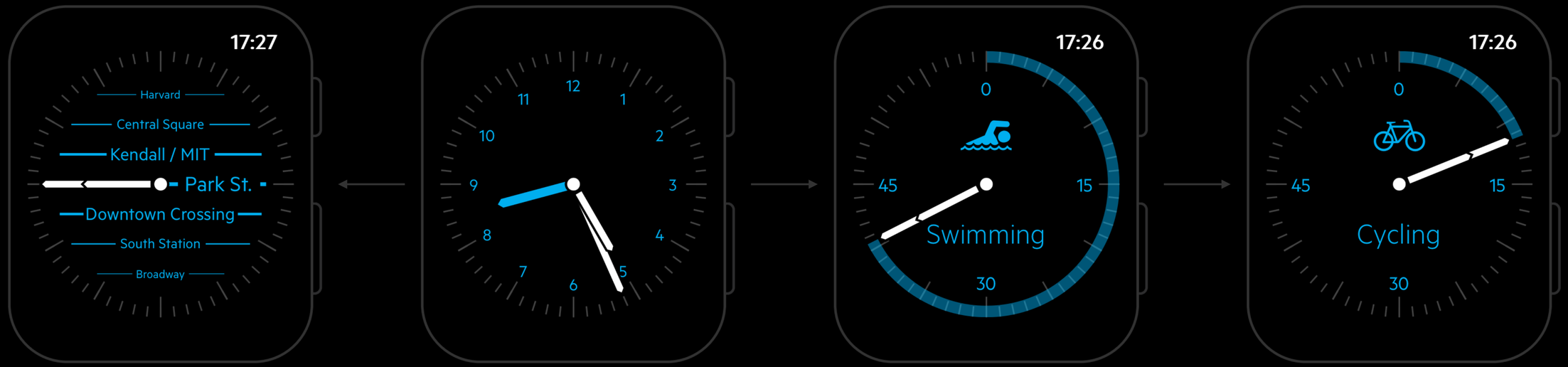
On mobile, I've hooked into the sensors to make twisting the bezel set timers, viewing the compass use the realtime magnetometer, and map mode react to rotation. On mobile, it also appears at 1:1 scale to help tune & design at the correct information density. You can take a look at the code itself as well on Github.



Cardinal

video

Cardinal



While not everything here would apply to a purely digital watch, automatic activity recognition is far enough along on the Apple Watch that it could now be used to create a new **context aware activity analog** watch face.

Tapping the display, like in other faces like **motion** could cycle between activities.

Case Study

Google Pixel Buds

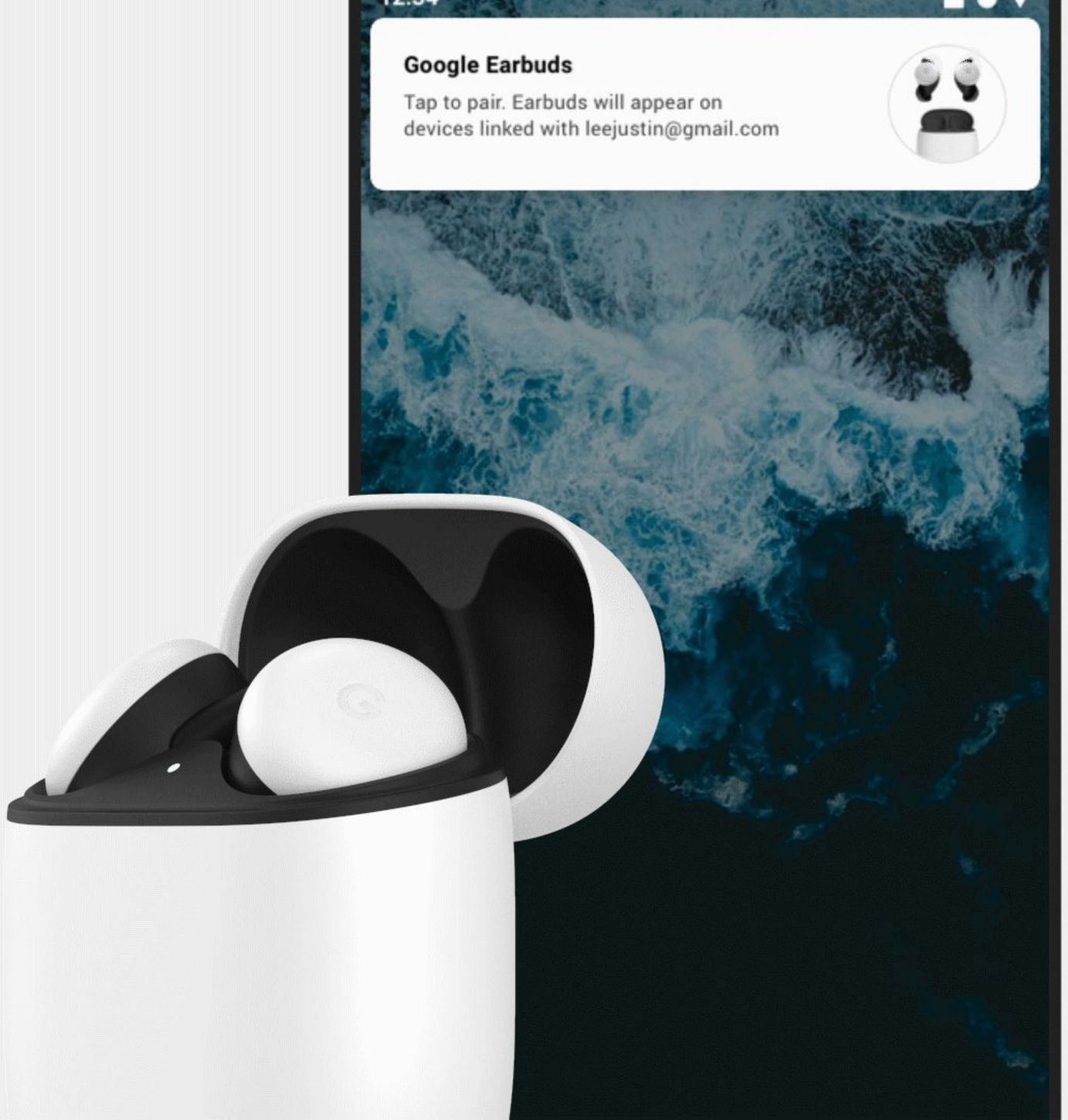
Google - In Progress

Pixel Buds

Similar to the paradigm Airpods popularized, we focused on having an extremely portable & easy to use charging case so that whenever you're not using your buds, they're always charging.

Pixel Buds

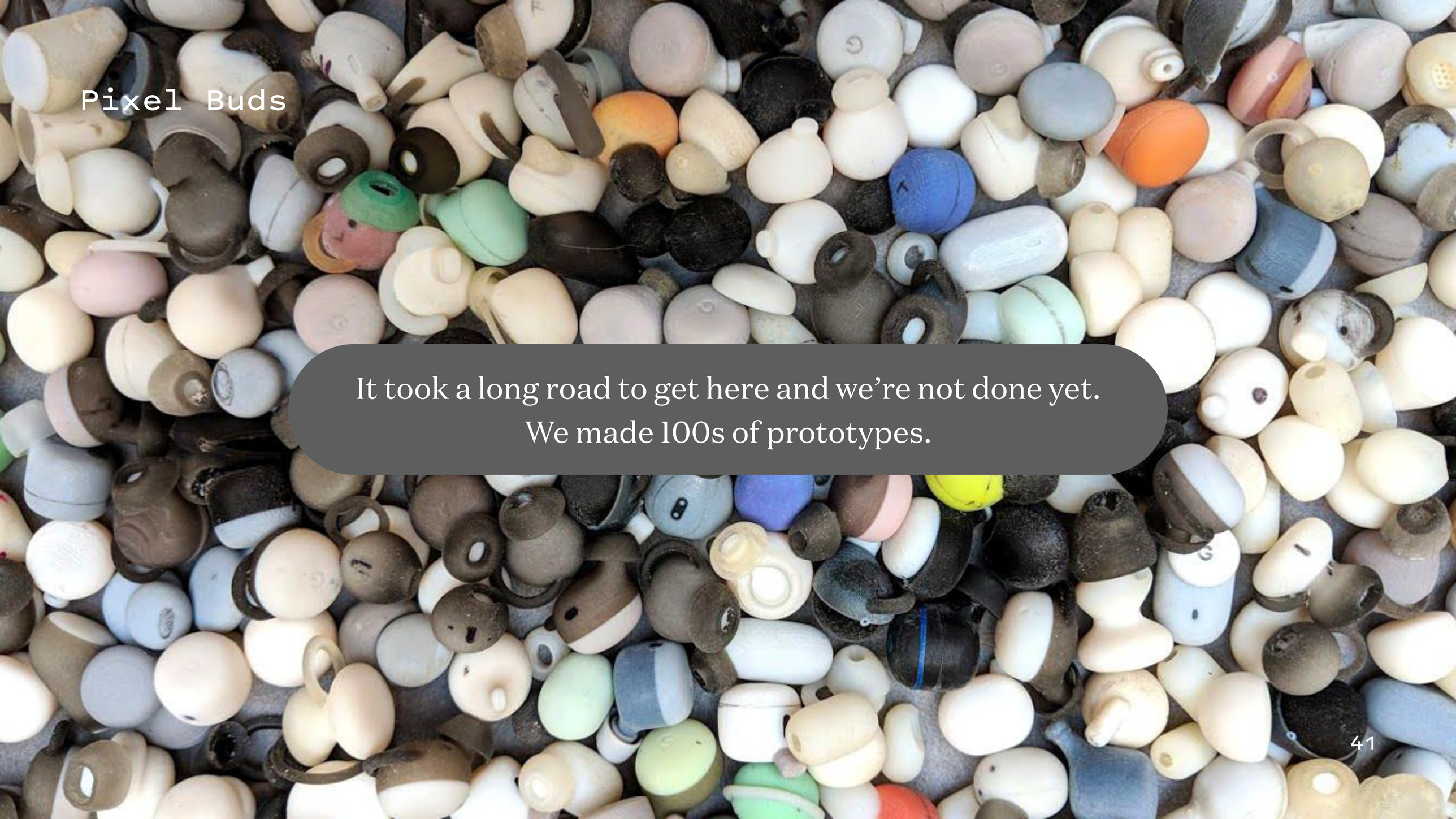
Just flip the lid and tap the screen to pair. With Fast Pair, pairing Presto happens with a flip of the case lid and a tapping a notification. Pixel users can be ready to go right away with all the controls integrated right in Settings.



Pixel Buds

Play when it's in, no missing out when it's out. Infrared sensors automatically detect when the bud is in your ear and pauses when you take it out.

Control your buds with just a tap or swipe. The capacitive touchpad in both earbuds so that you can use either hand or whether you're wearing one bud or two.



It took a long road to get here and we’re not done yet.
We made 100s of prototypes.


Pixel Buds
And placed them in 100s of ears.

2017

Sandeep
pm lead

Basheer
design lead

In late 2017, Sandeep and I saw a specific gap in our portfolio and began creating concepts for the project we lead now. Together, we used our bundled pitch of concepts & prototypes to convince our SVP of hardware to green light the project.

Once it had become official in spring 2018, we worked quickly to setup our new team and moved towards deepening our core roles as we gathered xx folks to build out the orgs outside of our expertise (like engineering, marketing, etc.)

2020

Fast forward to the present, and I now am a design lead for a team of eight across our own team and our partners.

Basheer
design lead

Alex
interaction designer

Erik
visual designer

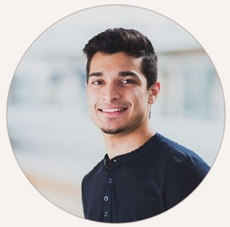
Frank
ux engineer

Nealeigh
ux writer

Alex
interaction designer

Justin
interaction designer

Shilp
associate ux



Chrisoula
interaction designer

Pixel Buds

Unlike our design teams' other programs, we have no sister team within Google that co-owns a core OS powering our product, so we own the complete UX from end to end. This covers:

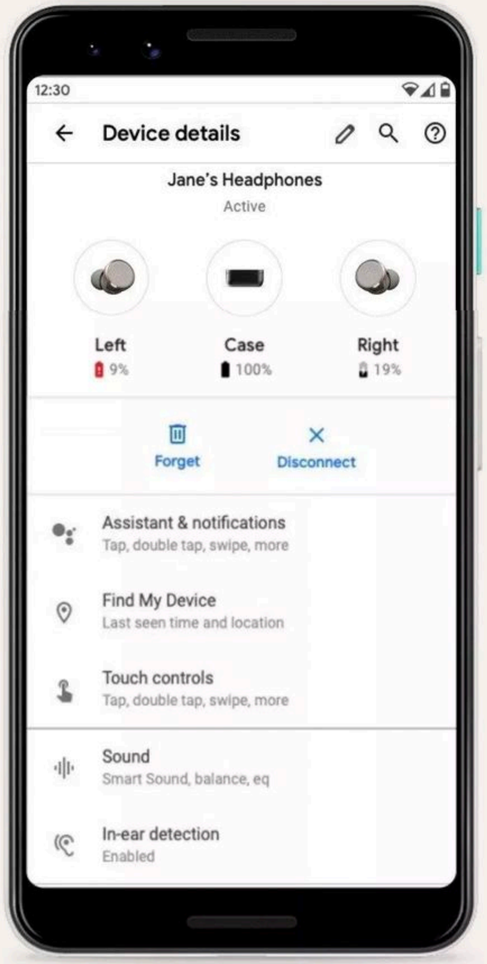
hardware		software	
buttons	sound design	companion app	android integration
lights	battery & charging	onboarding	android bluetooth
input & gestures	kpi's	find my device	fast pair
connectivity & power	quick start guide	notifications	assistant integration
volume control	packaging	help center	app integrations

Android Q

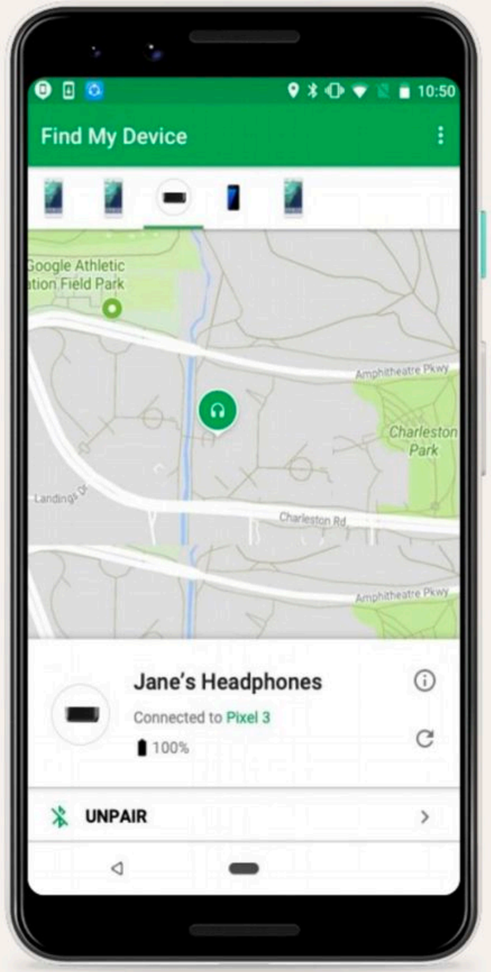
While the full hardware product is still unreleased, some of the software features we've been working on have already started shipping in the public beta versions of the next version of Android.

The final visual design has not been implemented yet.

fast pair 2.0



newly expanded bluetooth settings



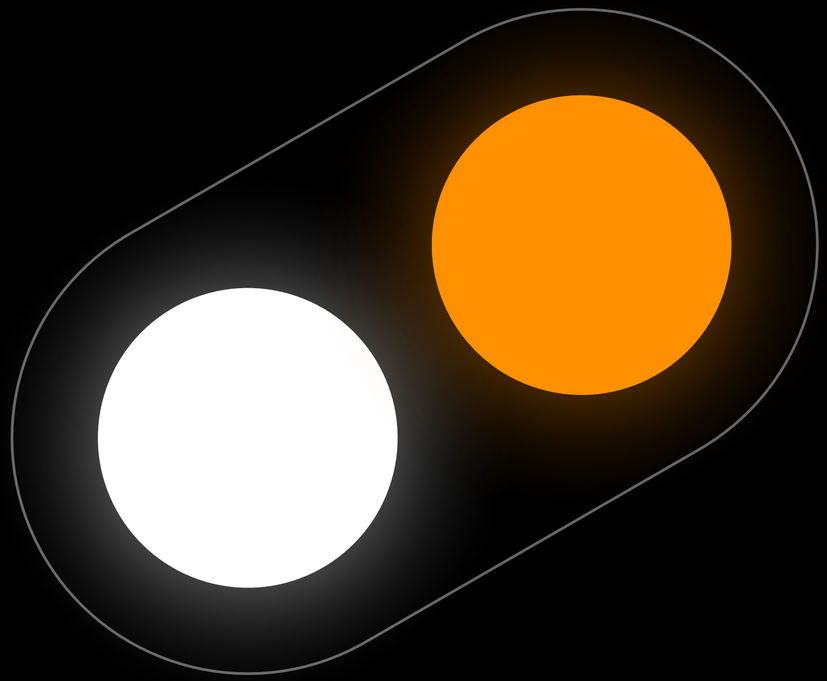
find my device

Lights

Light Anatomy

A single light pipe / opening, each containing
2 discrete diodes (separate or single package): 1 white + 1 amber

colors	After shining through CMF: • white, 4000K - 5000K, or #FFFFFF • amber, 590nm ± 10, or #FF9100
contrast	should be somewhat visible in off state
driver	able to smoothly fade the LEDs on a sine curve without noticeable steps or drop-off
concurrency	no ux requirement to be able to run or mix both LEDs at the same time
integration	able to run LEDs with full capabilities in all major power states of the device, excluding ship mode
brightness	~10mcd through a combination of initial specs & current-limitation in the driver without clamping PWM range
viewing angle	able to tell on vs. off within 160°



Front Light

All plugged states take precedence over the unplugged states. Ship mode negates all LED states.

light state	unplugged	charging
white	battery: > ~1 full charge left, 30s timeout	battery full
amber		charging
blinking amber	critically low battery: < ~1 full charge left, 30s timeout	
off	not enough power to charge anymore	

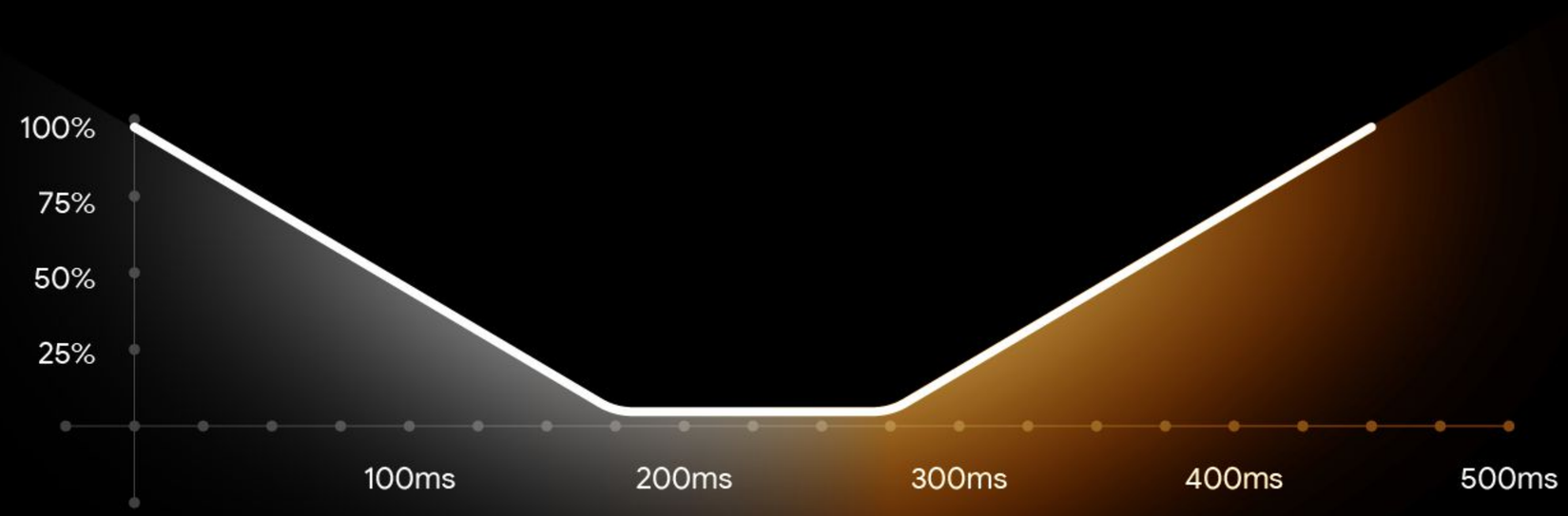
Top Light

This light is both only on and only seen while the device is open, with an x timeout largely unnoticed by users.

light state	docked	undocked	timeout
white	batteries are full		30s
bounce white	in pairing mode		As long as in pairing mode
amber	charging		30s
white ↔ amber	factory reset in progress		As long as in reset, and on for minimum 2.4s
error blink		missing, can not pair / factory reset	Play once 1.2s, then stay off until button released
off	battery so low it can't charge anymore	not charging	

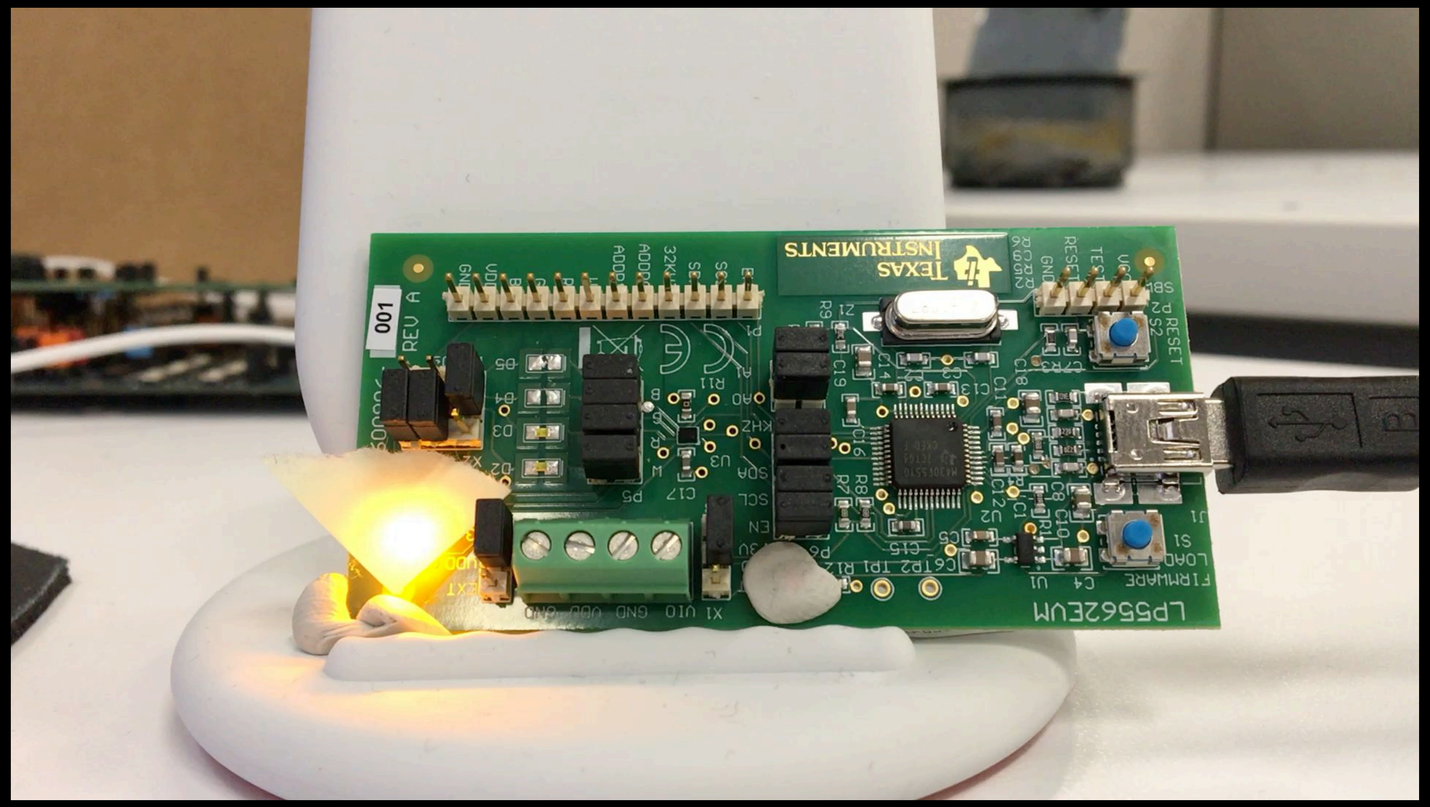
Light Transition

When the light is switching from showing one system state to another, it should always perform this transition, even if the starting and end light colors are identical



Light Transition

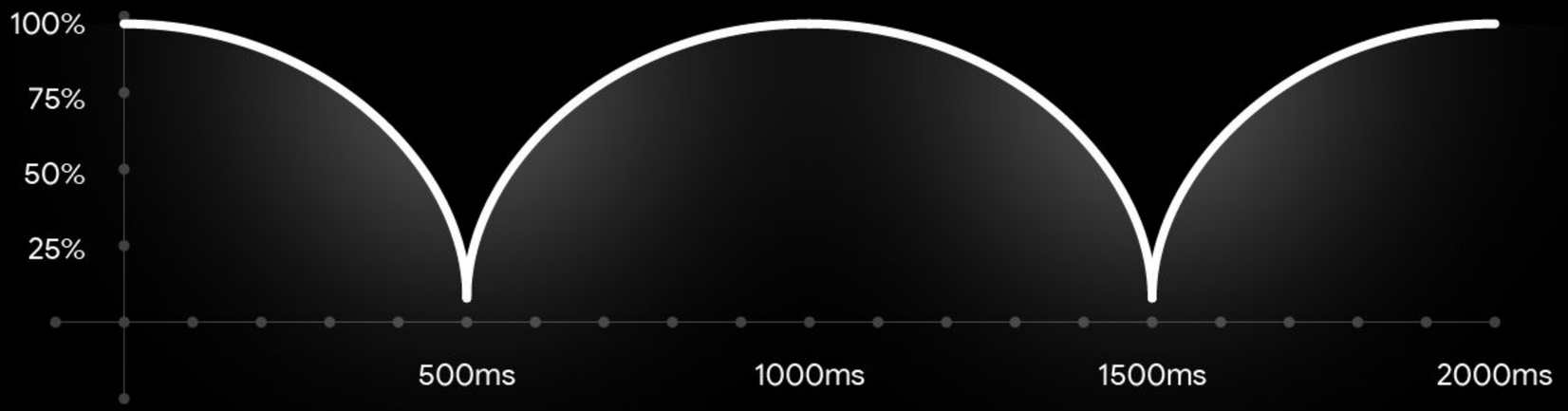
When the light is switching from showing one system state to another, it should always perform this transition, even if the starting and end light colors are identical




```
# Spec: Ramp between two
states (two engines/colors)

.ENGINE1            .ENGINE2
set_pwm 0           trigger w1
ramp 87.5, 128      set_pwm 0
ramp 87.5, 128      ramp 87.5, 128
wait 999            ramp 87.5, 128
wait 500            wait 999
ramp 87.5, -128     wait 500
ramp 87.5, -128     ramp 87.5, -128
wait 100            ramp 87.5, -128
trigger s2          wait 100
trigger w2          trigger s1
start               start
```

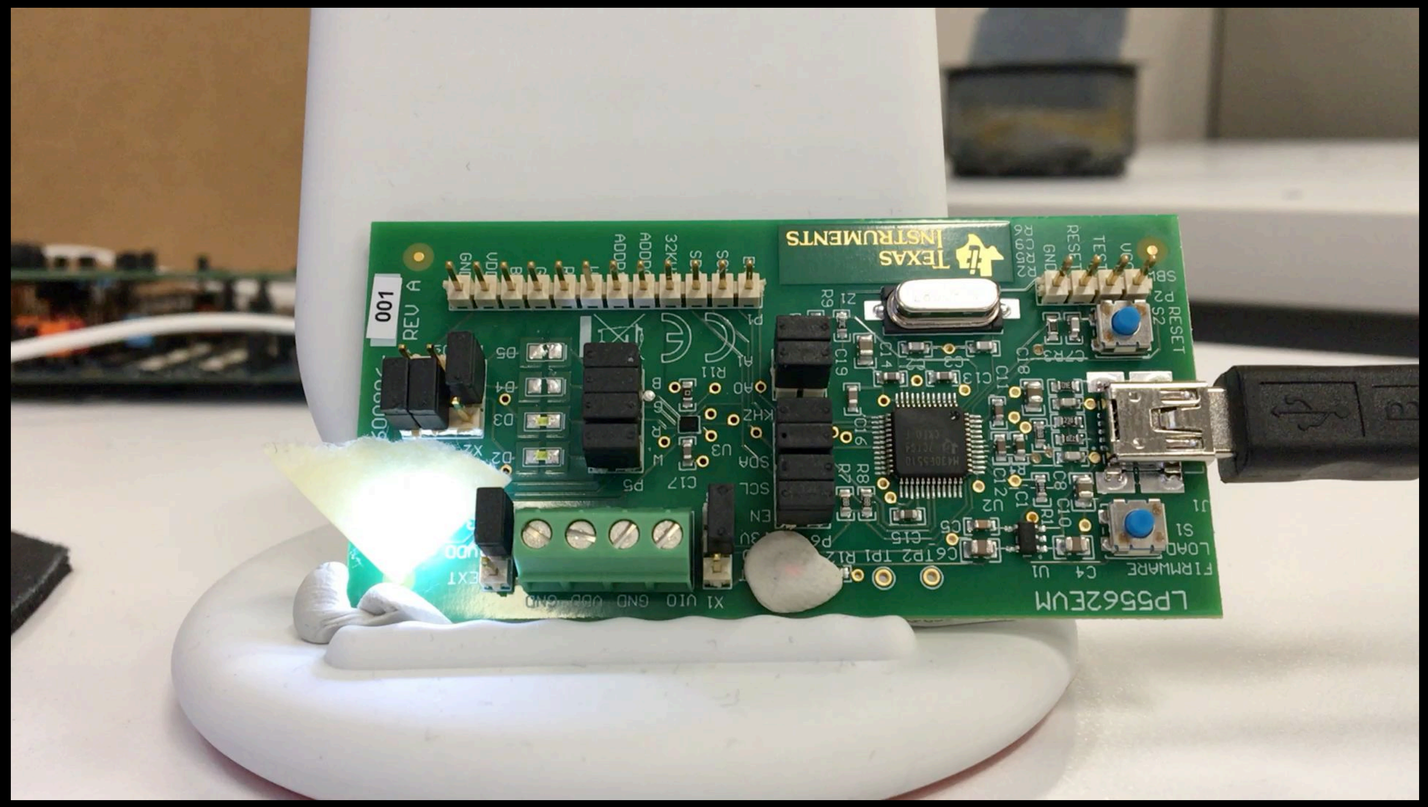
Bouncing White

The general curve matches an absolute value sine curve



Bouncing White

The general curve matches an absolute value sine curve



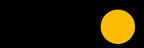

```
# Spec: Brightness follows
linearized absolute value
sine wave with 1s period

.ENGINE1
set_pwm 0
ramp 155, 124
ramp 170, 102
ramp 145, 29
wait 60
ramp 145, -29
ramp 170, -102
ramp 155, -124
start
```

Pixel Buds

To test our spec early in the product development process, and to help lock the decision on the # of LEDs, I made an interactive physical prototype to use in a user study that enabled the real spec to be implemented at a scale & hackiness that we could achieve alone.

Touchpad

Pixel Buds



Play Pause / Answer Call

Tap

Next Track / End Call / Reject Call

Double tap

Previous Track

Triple tap

Google Assistant / Check Notifications

Push to talk (only after OOBE)

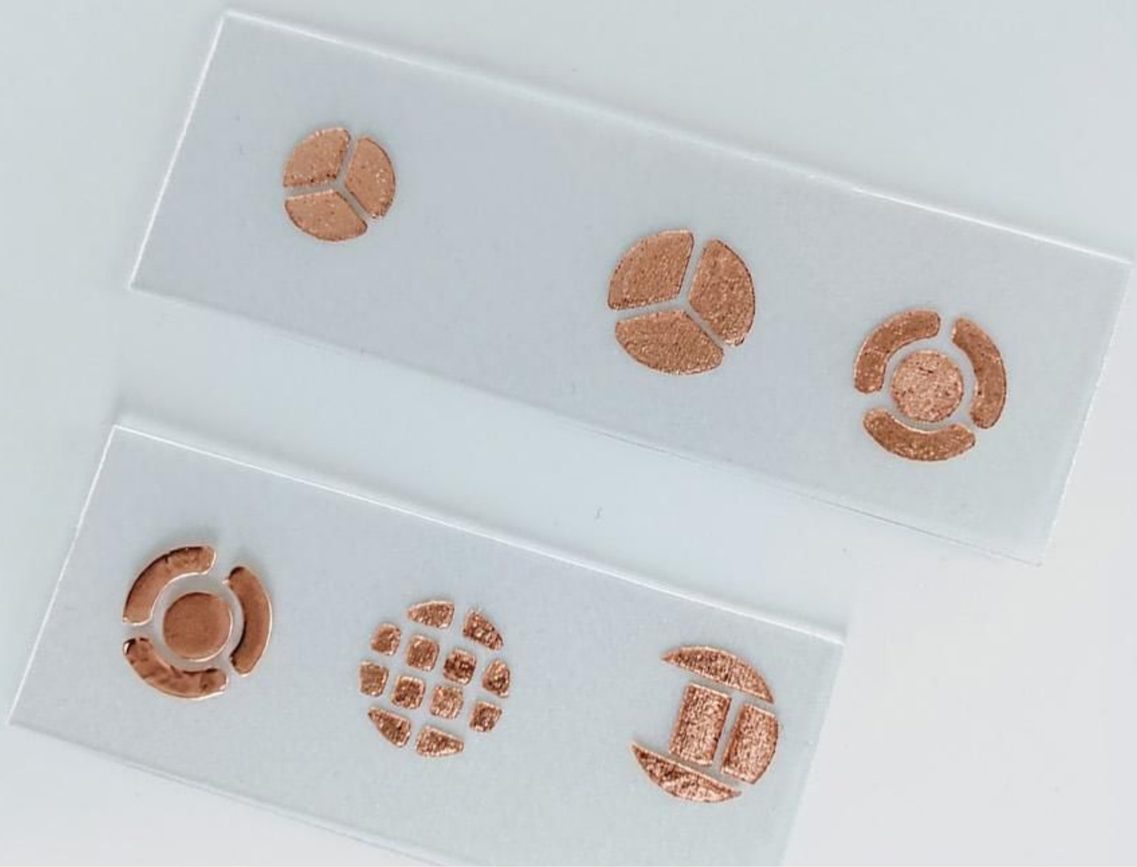
Raise / Lower volume

Swipe forward / backward

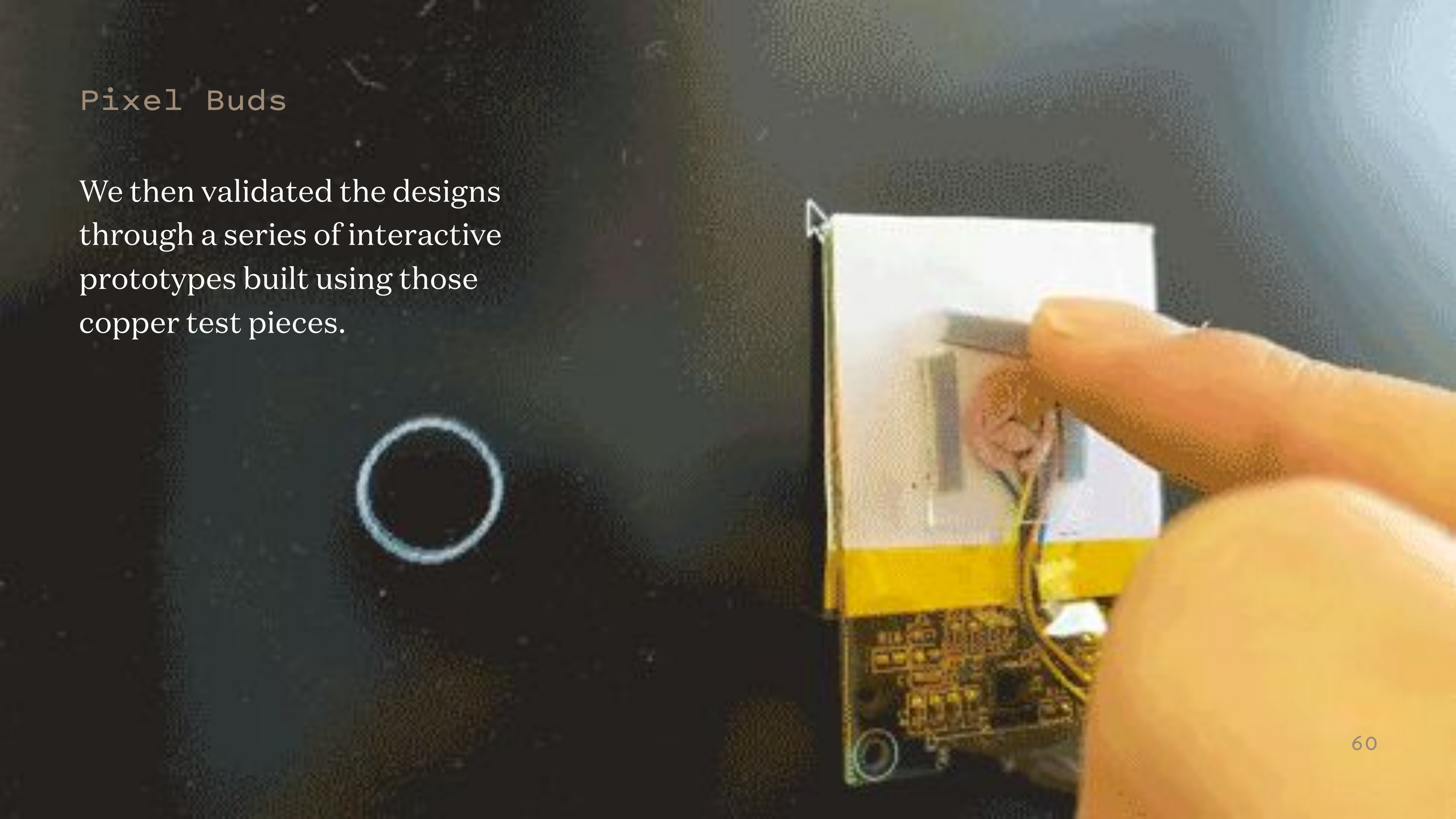
Pixel Buds

gestures	music / idle	phone call / ringing	assistant	translate
	play / pause	pickup	n/a	toggle mute
	next	reject / hangup	dismiss	
	previous			
	assistant / fetch	n/a	reply / new query	push to talk
	volume down			
	volume up			

To test our spec early in the product development process, and to help lock the decision on the touchpad, I used our vinyl cutting machine with copper tape to quickly iterate on patterns alongside our UXE.



We then validated the designs through a series of interactive prototypes built using those copper test pieces.

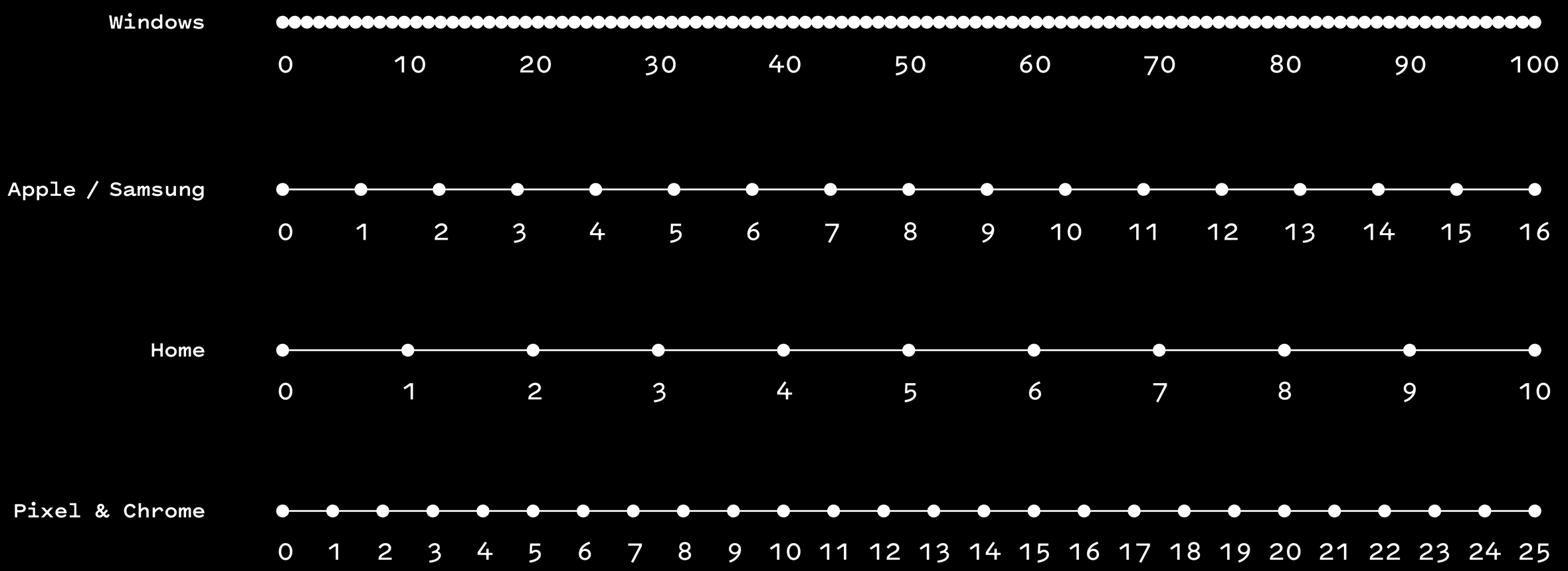


Volume

Principles

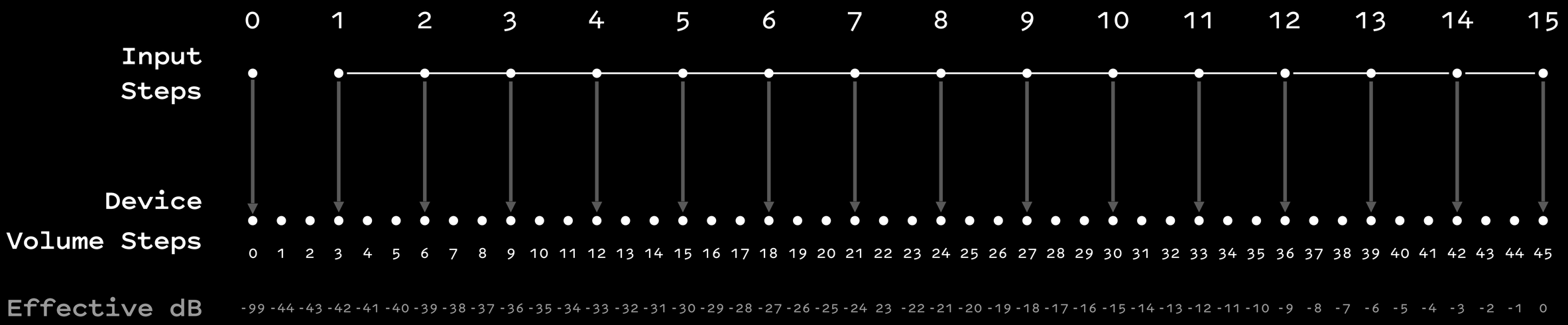
`input`	Users can adjust the volume via the host device, the Assistant, or the device.
`snapping`	Steps are snapped to the nearest step of the **Device Volume Steps (45+1)**, not the input steps nor the host's steps.
`initial setting`	• Most host devices already manage & remember the starting volume level. For Android, this was implemented in Pie. • Until overridden by the host, **preserve the last used (current) volume level**. • When that value is unknown the default should be set to ~50%.
`steps`	We've chosen **15 audible steps + a special 0-media-volume state via input**. In principle, each step up/down in volume should not feel too abrupt (human ear perception threshold is about 3dba on average). Users should feel in control of their volume - not too big or small of a jump. This should strike a balance between users not having to input too often to increase/decrease volume and feeling in control of their volume.
`minimum`	UI sounds and phone calls should never be able to be set to zero volume. However, media playback should.

Volume Steps across Host Products



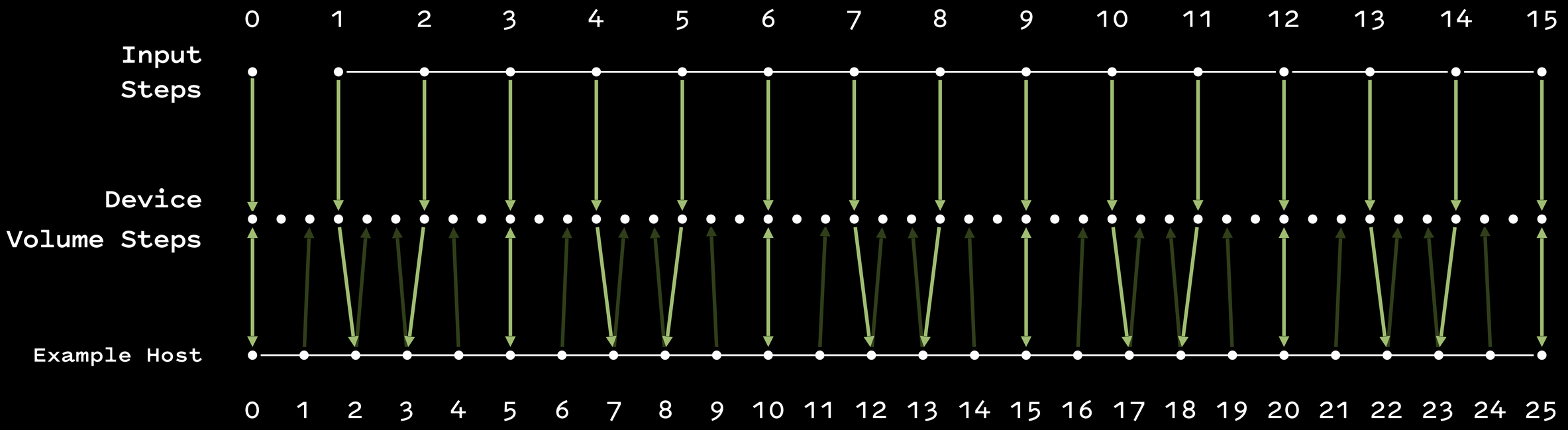
Device Volume Steps

We will use our own 15+1 step scale for input, but report a more granular scale (45+1) as an individual possible step.
Essentially, the input “skips” multiple device volume steps rather than having # of volume steps be equal to # of inputs.



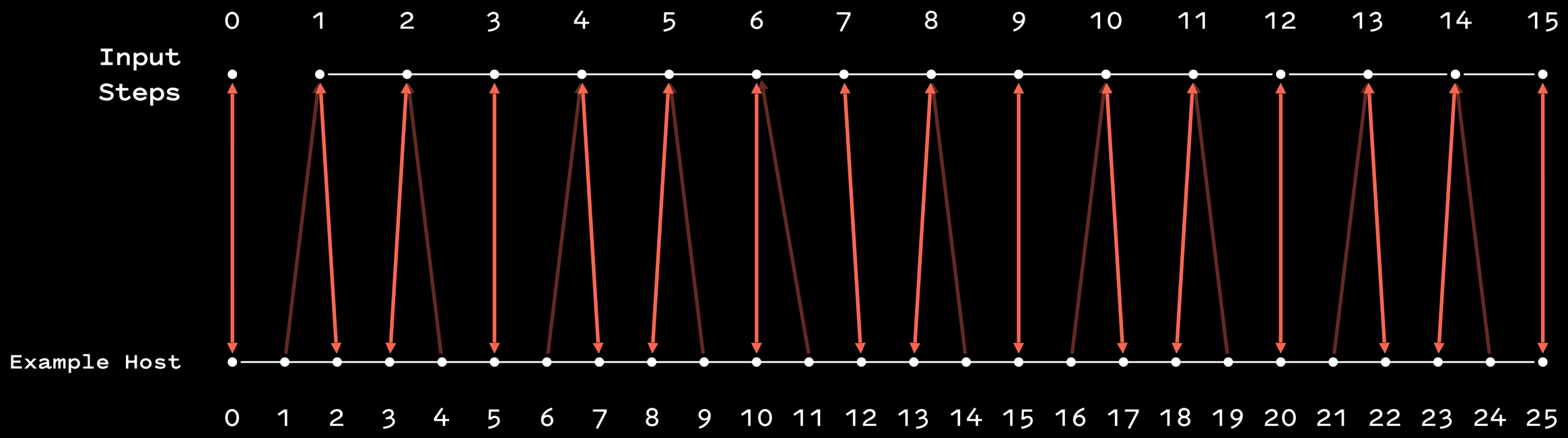
Example Mapping: Correct

Input on the device snaps to the nearest device volume step. Changing the volume on the host snaps to the nearest device volume step as well.
This means that a host (like the Pixel phone) with volume steps more granular than our input will be able to set a volume in-between that of the input.



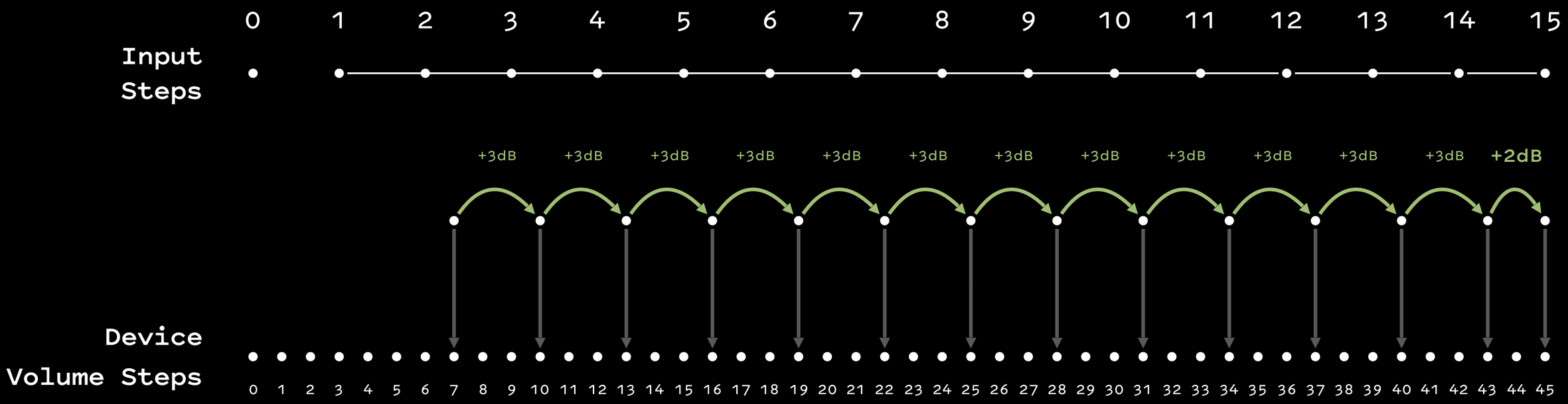
Example Mapping: Incorrect

An incorrect interpretation of this would limit the granularity of the host's volume steps to the very coarse setting we've picked for our input. Performing many repeated inputs on the device is tiresome, more than a touchscreen, which is why we've limited it that way.



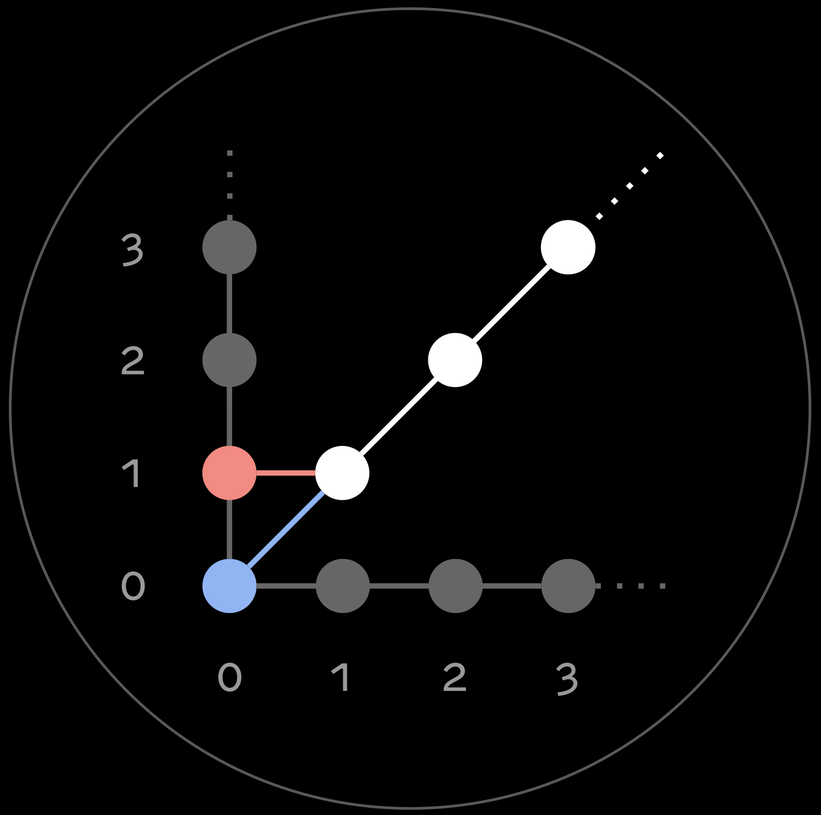
Relative Volume Change Behavior

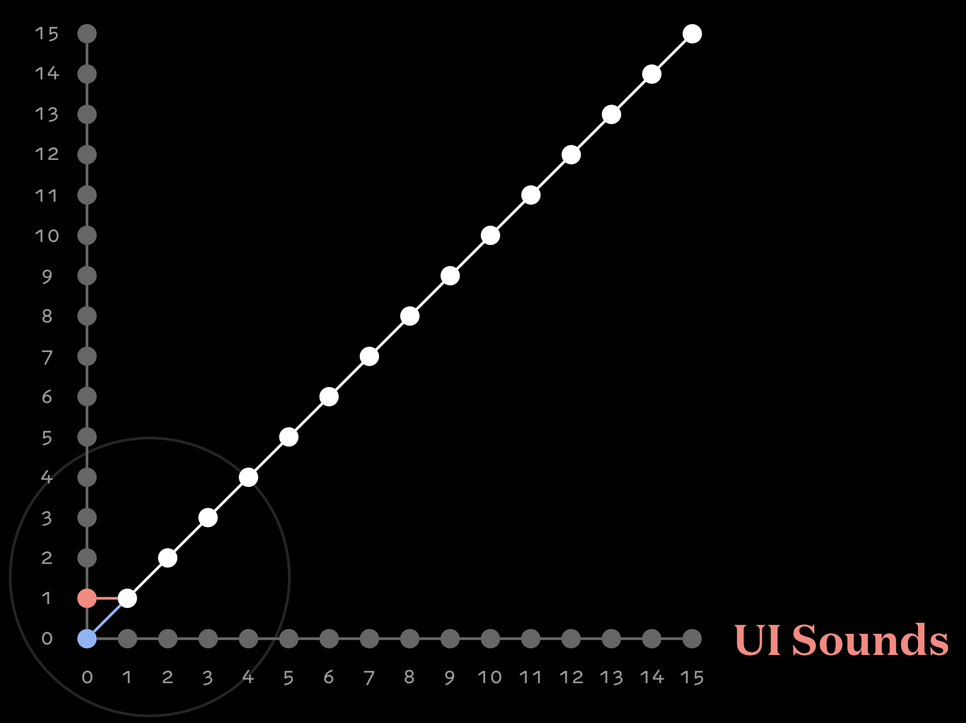
When the user changes the volume, the device should always jump +3 to the nearest device volumes step, not the nearest input step. If the host device was used to set a volume level in-between the input steps, that same in-between offset would be maintained unless they hit the min or max volume.



UI Sound Volume: Media

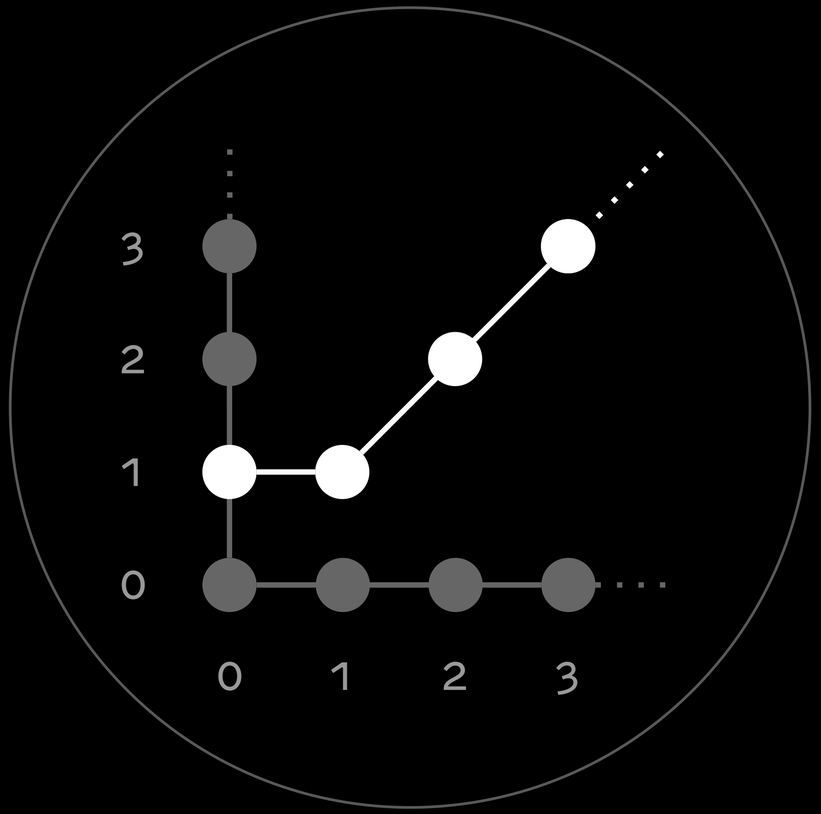
The volume of the ui sounds during media playback (A2DP) and how they map to the volume of media. Generally, the relationship is 1:1 until volume zero where media is allowed to reach zero but ui sounds are not.

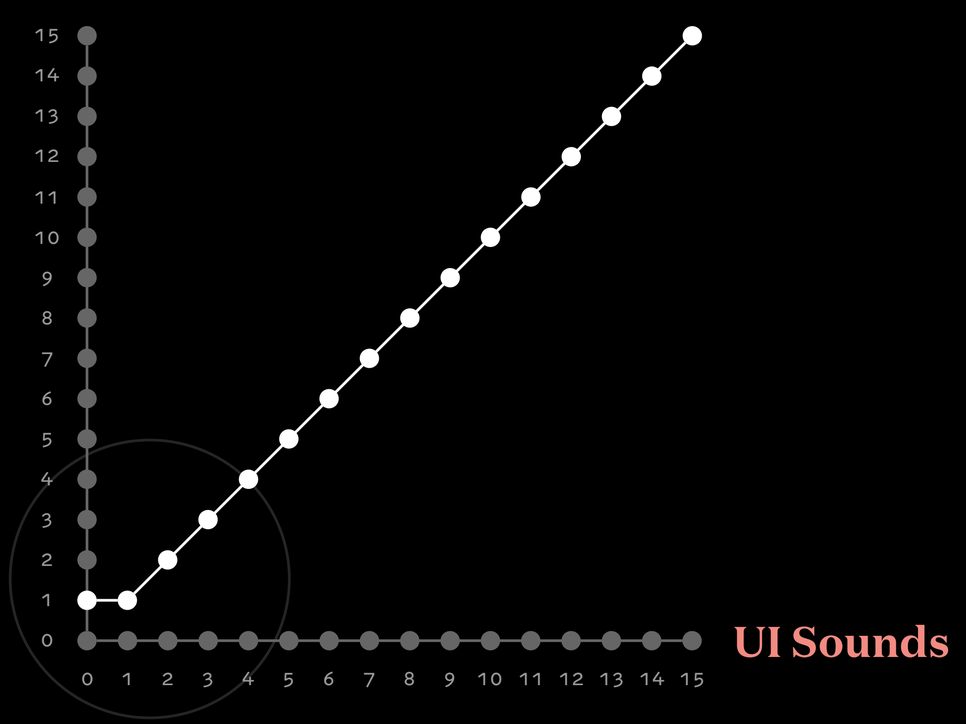




UI Sound Volume: Calls

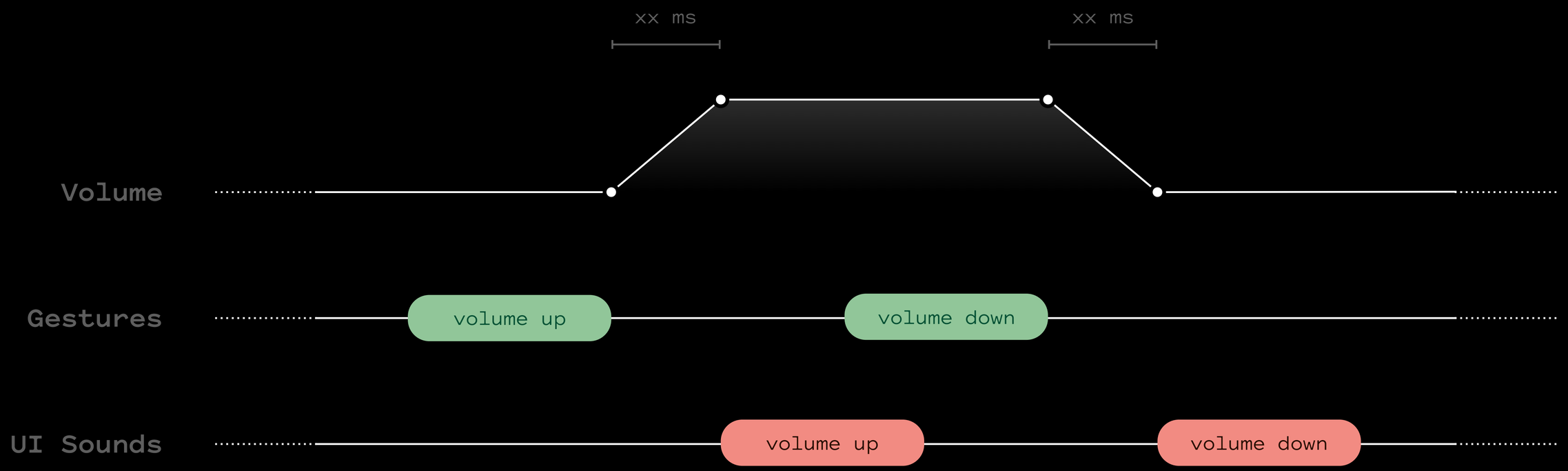
The volume of the ui sounds during phone calls (HFP) and how they map to the volume of calls. Generally, the relationship is 1:1 until volume zero, where **both** are unable to reach actual zero volume.



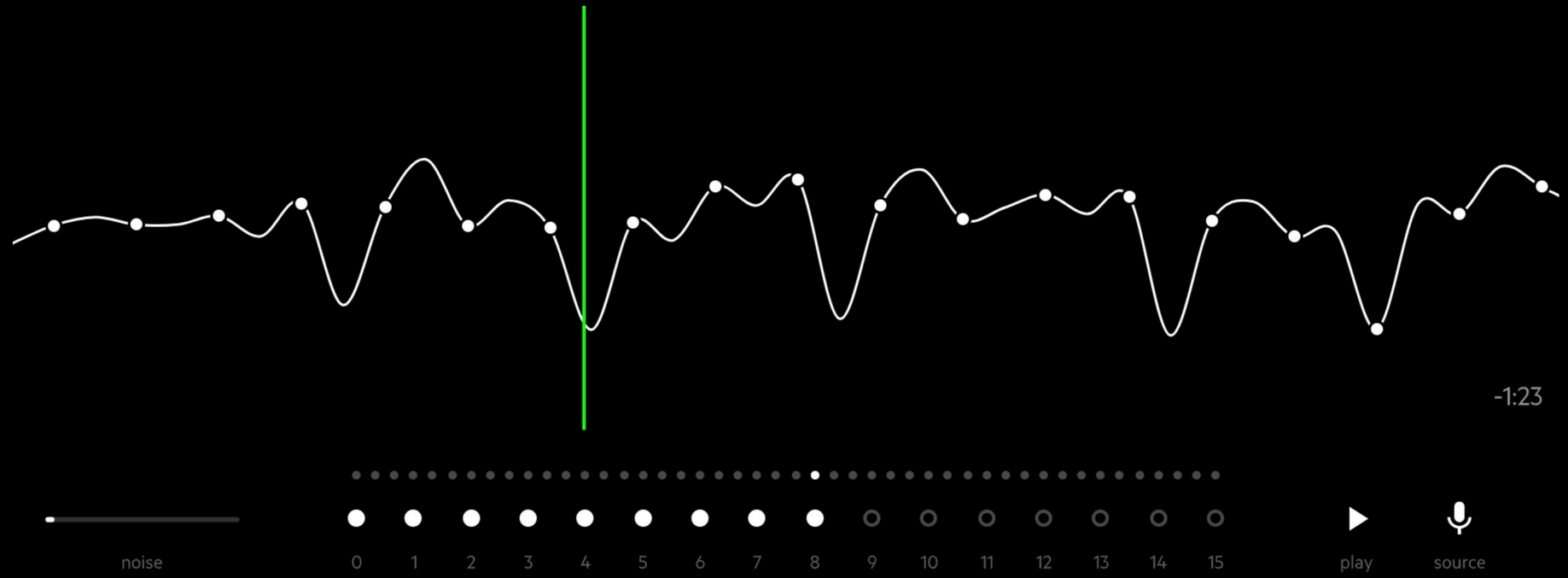


Volume Change Transition

To prevent glitching & popping of the audio, there should be a small linear ramp transition between each volume change. The ui sounds should then always subsequently played once the ramp is complete. The ramp time (xx ms) is likely defined by the audio driver.



Pixel Buds



To quickly explore and iterate on the interactions, I started building a prototype in javascript with p5js.

basheer.co/sound

Thanks!

basheer@basheertome.com