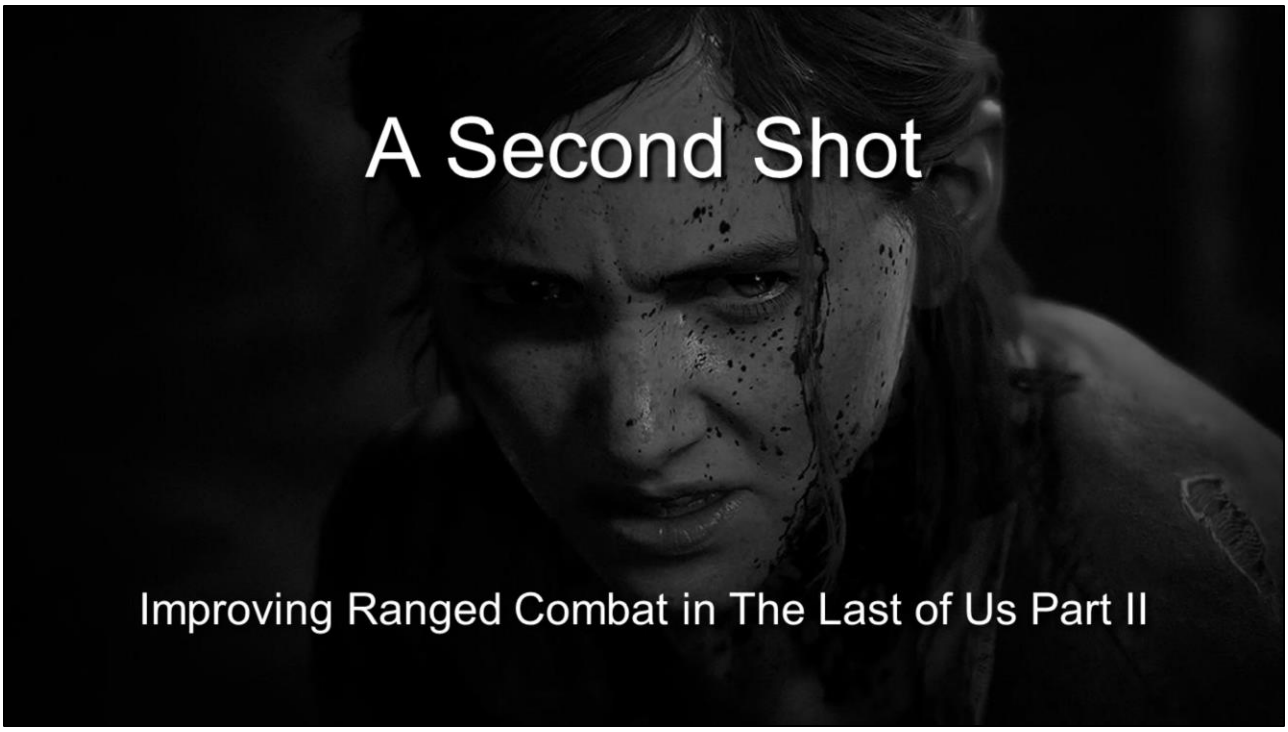


Welcome to talk.

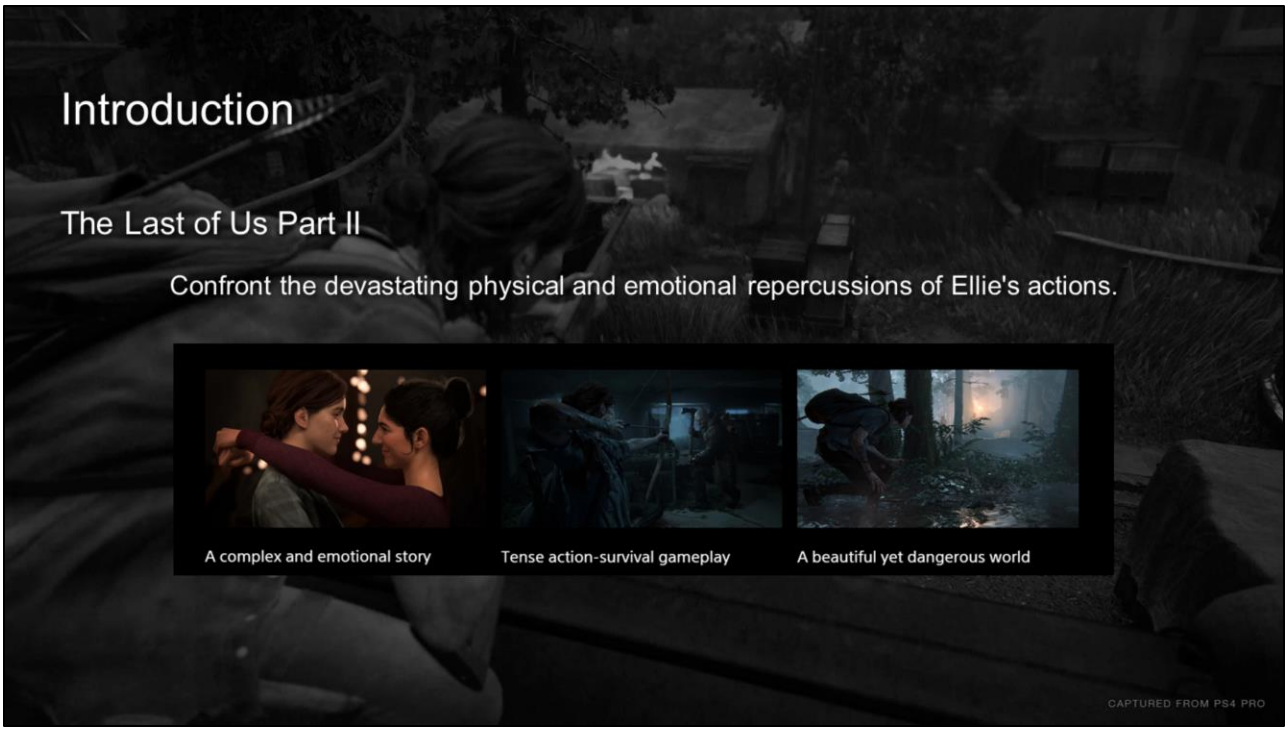
Title Page

Warning!

In case you aren’t familiar with TLOU series, it is rated MA for Mature.

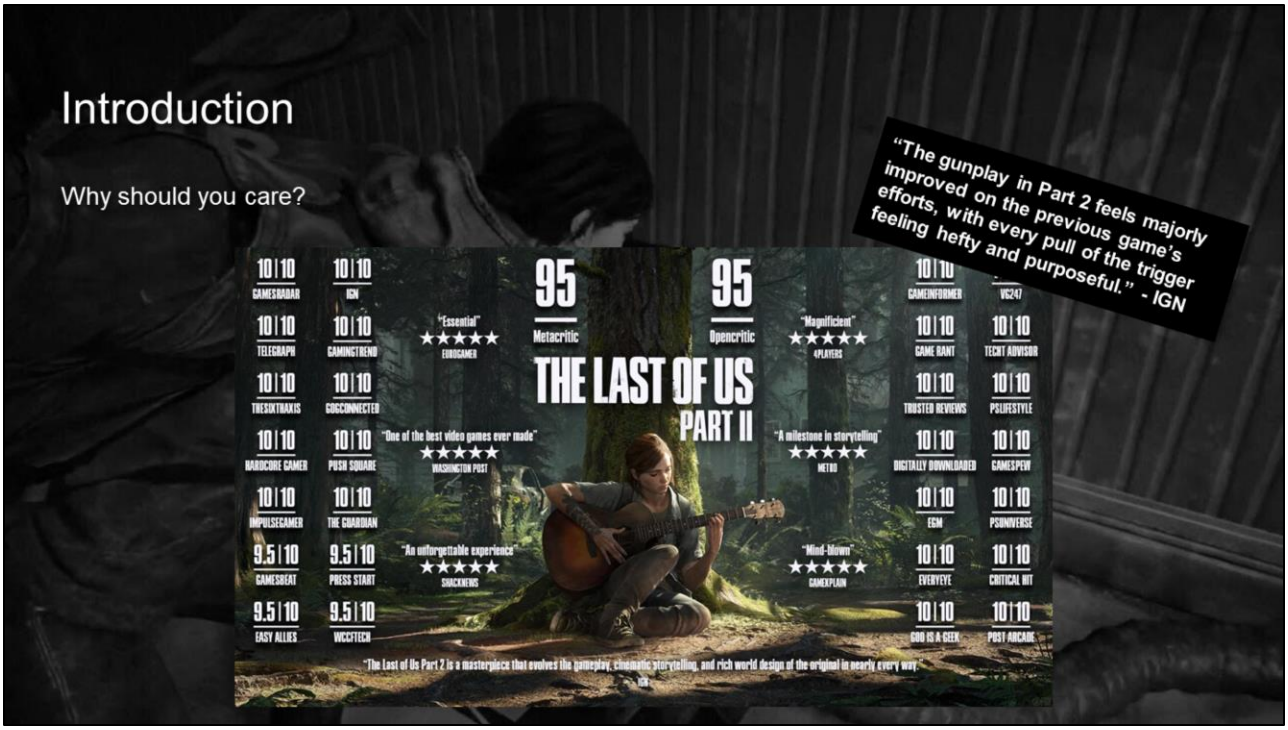
As a result, this talk will have Graphic Content in it: Most notably violence and gore.

If you are uncomfortable with this feel free to hop out, I won’t take offense at all, also we are ~10ish slides out from any of that content so you have time.



Also, in case you haven't played or are aware of TLOU2 here is a brief overview.

Without delving too much into the narrative, TLOU2 is a game set in a post pandemic world, where we explore the complex aspects of a relentless pursuit for vengeance through tense action-survival gameplay.



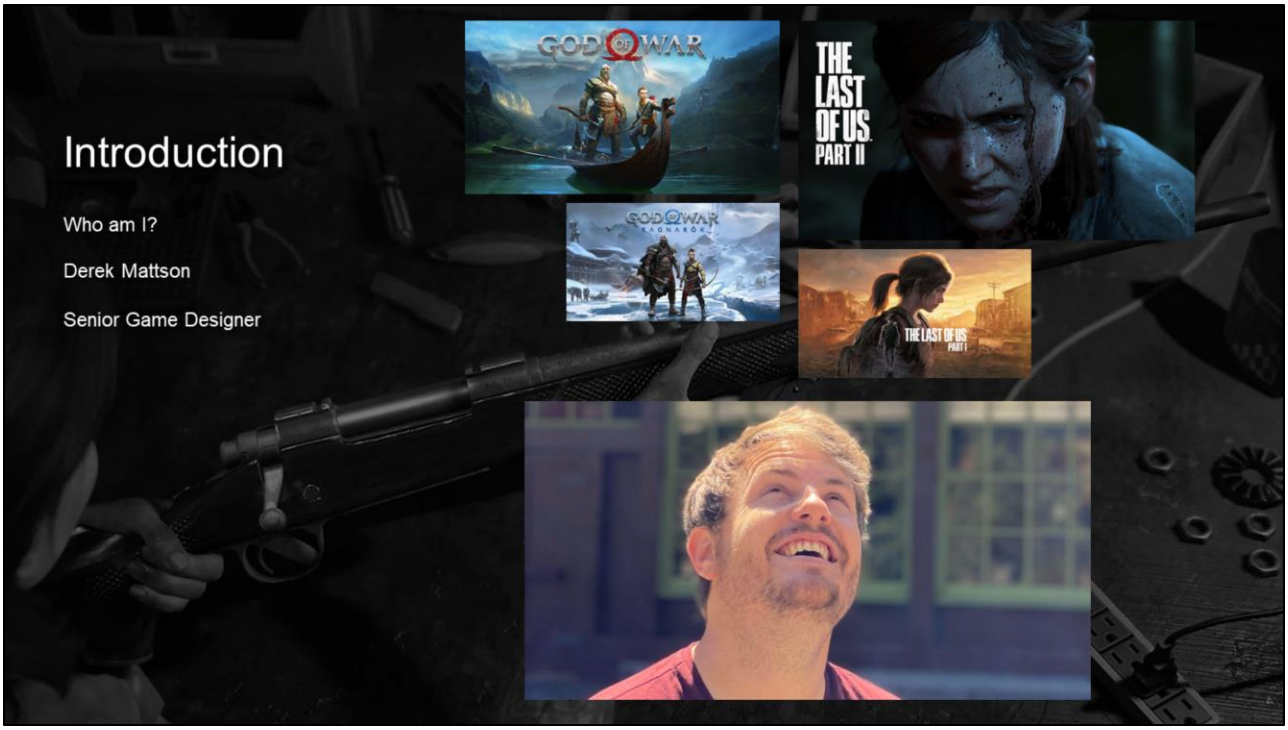
Why should you care about this talk?

The game was critically acclaimed on release, with even its harshest of critics often still noting gameplay improvements.

(click)

And here is a little quote about the ranged combat that I was particularly proud of.

“The gunplay in Part 2 feels majorly improved on the previous game’s efforts, with every pull of the trigger feeling hefty and purposeful.” - IGN
https://www.ign.com/articles/the-last-of-us-part-2s-combat-cycle-beating-hearts-and-butchery



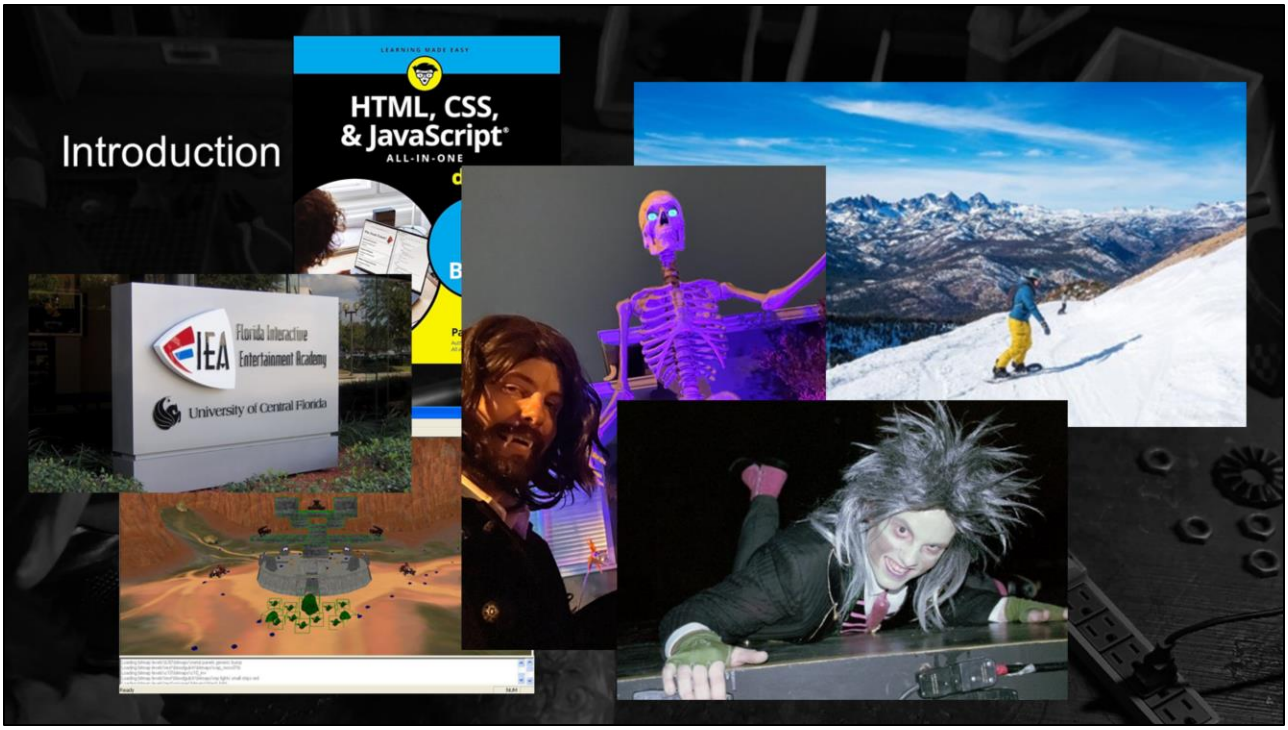
Who am I?

(click)

My name is Derek Mattson,

(click)

I am a Senior Technical Game Designer at Naughty Dog. I've been there just over 5 years and been with Sony for about 8, and I was the design owner for Player Weapons on The Last of Us Part II.



But deep down, (click) I'm just a theater kid (click) who grew up in Florida, taught himself to script (click), modded halo combat evolved levels growing up, (click) and then went to grad school for Interactive Entertainment.

(click) And now I just use Halloween as an outlet for my weird theater side, and (click) was able to learn about snow by leaving the state of Florida.

I want to go over some terminology that is commonly used in discussion of shooting mechanics.

Projectile - This refers to weapons who's probe travels over time, (click) continuing to probe every frame until it collides with something.

Hitscan - This refers to weapons that probe for 1 frame, (click) everything else that lingers beyond that is just feedback and does not directly contribute to whether or not a hit lands.

The 3 Topics I am going to dive into today on improving ranged combat, cover the changes that I think had the largest impact.

Projectiles
“Weapon Feel” - In quotes for a reason that I will dig into later
Hit Scan

(click)

Starting with projectiles

In the original TLOU there was a Bow system that handled more closely to a grenade than a ranged weapon.

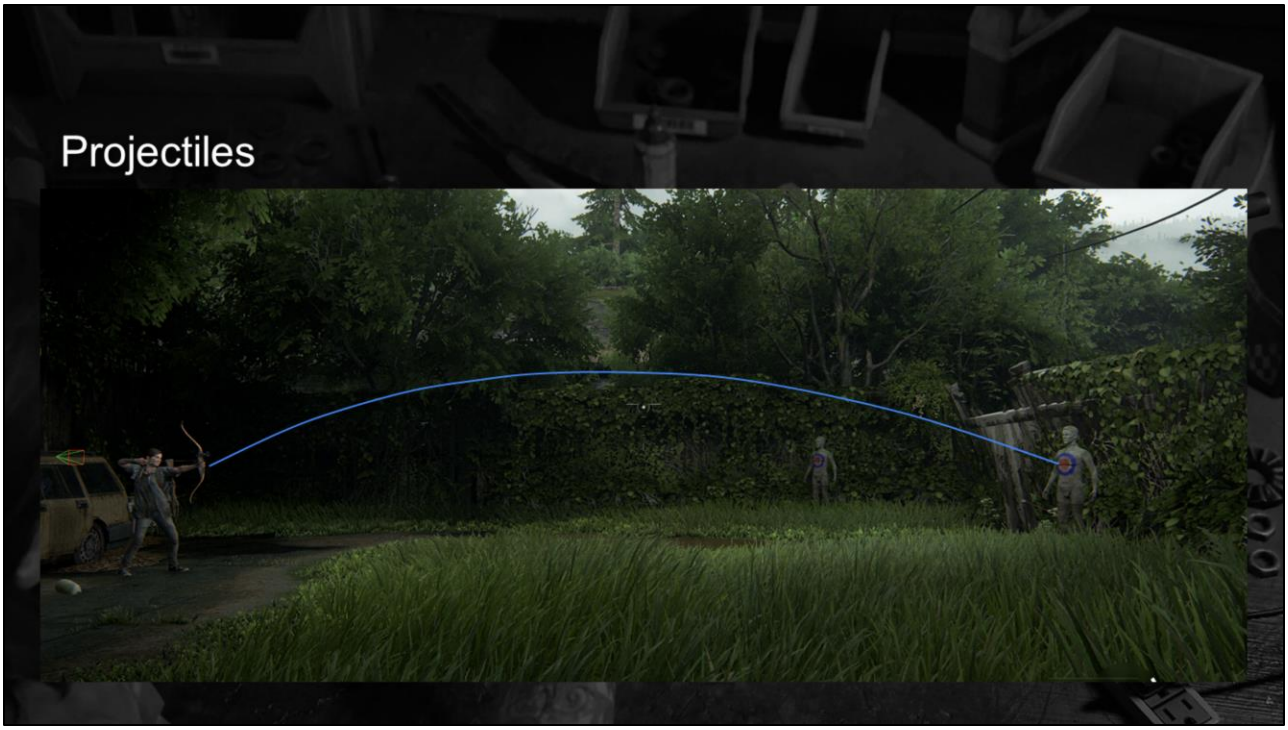
There was an arc trajectory you could preview and a landing location for where it would explicitly collide.

Directors knew early on they wanted to change this and bring the weapon closer in line with how our other ranged weapons handled.

Next Slide

Here is a screenshot of the game as it shipped.

Ignoring the reticle, would you be able to tell me where the arrow is going to go?
(click)
Does it go straight?
(click)
Does it arc up and lower over time?

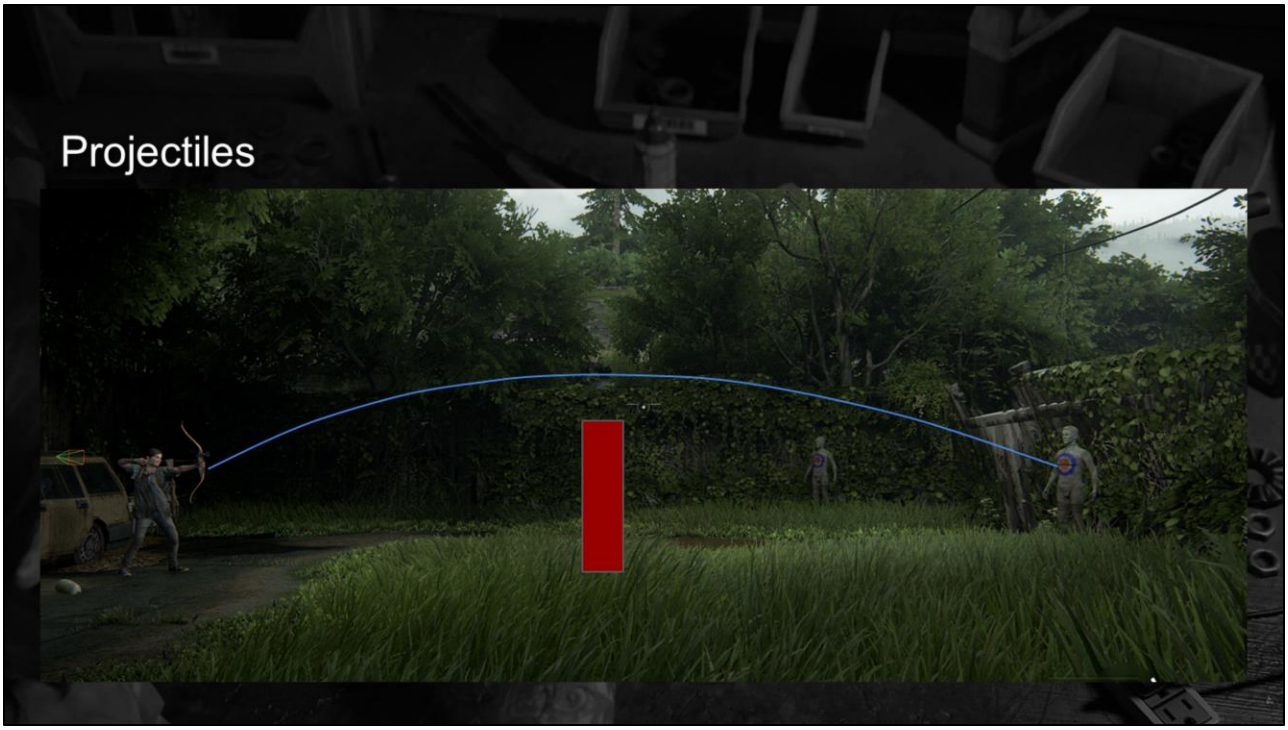


Let’s look at a side view of this exact same shot.

In the previous system, which used a directional vector, a flat velocity, and flat gravity value, we probably would have tilted the vector up a bit to land the shot at 10 meters in order to prevent the projectile from having to move at extreme velocities.

But what happens if that target is closer than say 5m?

Next Slide



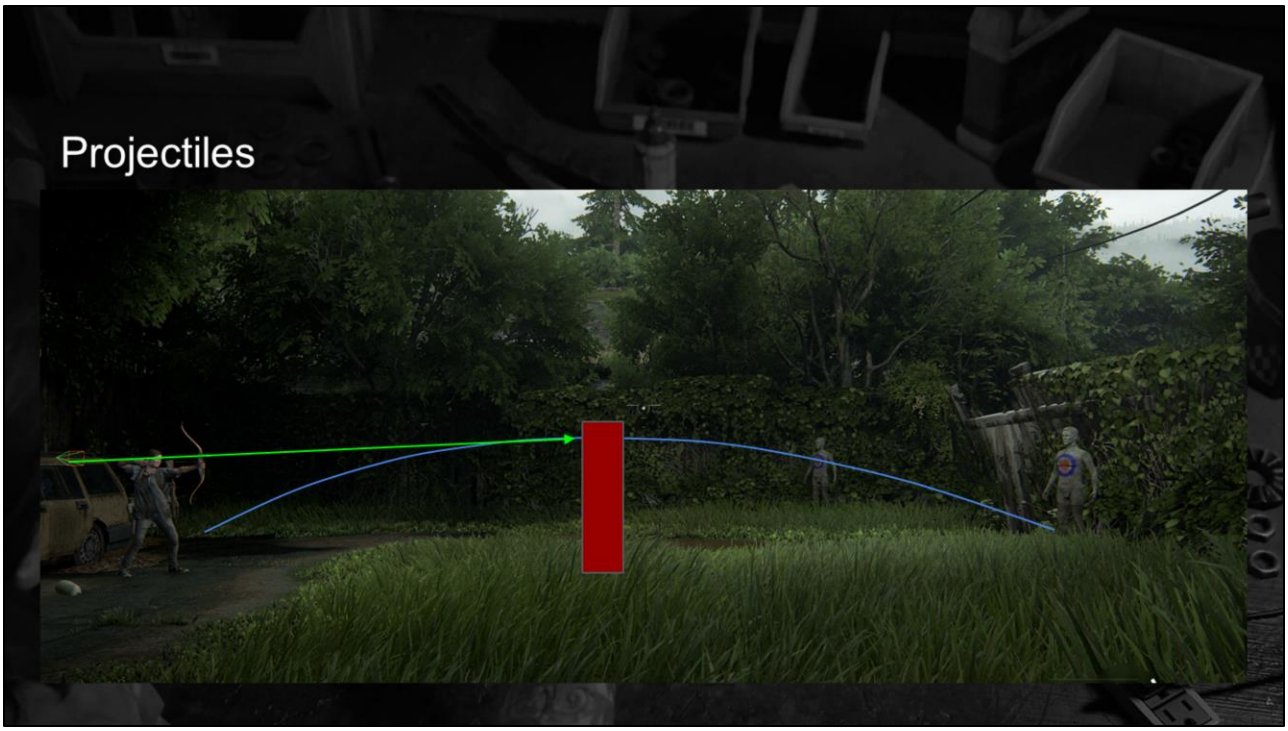
Well then we would over shoot them right?

But we can’t lower the directional vector anymore without losing our 10m target.

Okay, well what if we lower the curve and change the spawn location?

Well no that doesn't feel good either.

Also, does the arrow spawn from my feet now?



With a 2D reticle, what we really kind of end up wanting is for that first 5m or so pretty much not be a part of the parabolic curve.

Here is the debug for the in game projectile prediction that we ended up with.

You'll notice it is quite different than a traditional parabolic curve.

And that it lands pretty consistently for those first few frames.

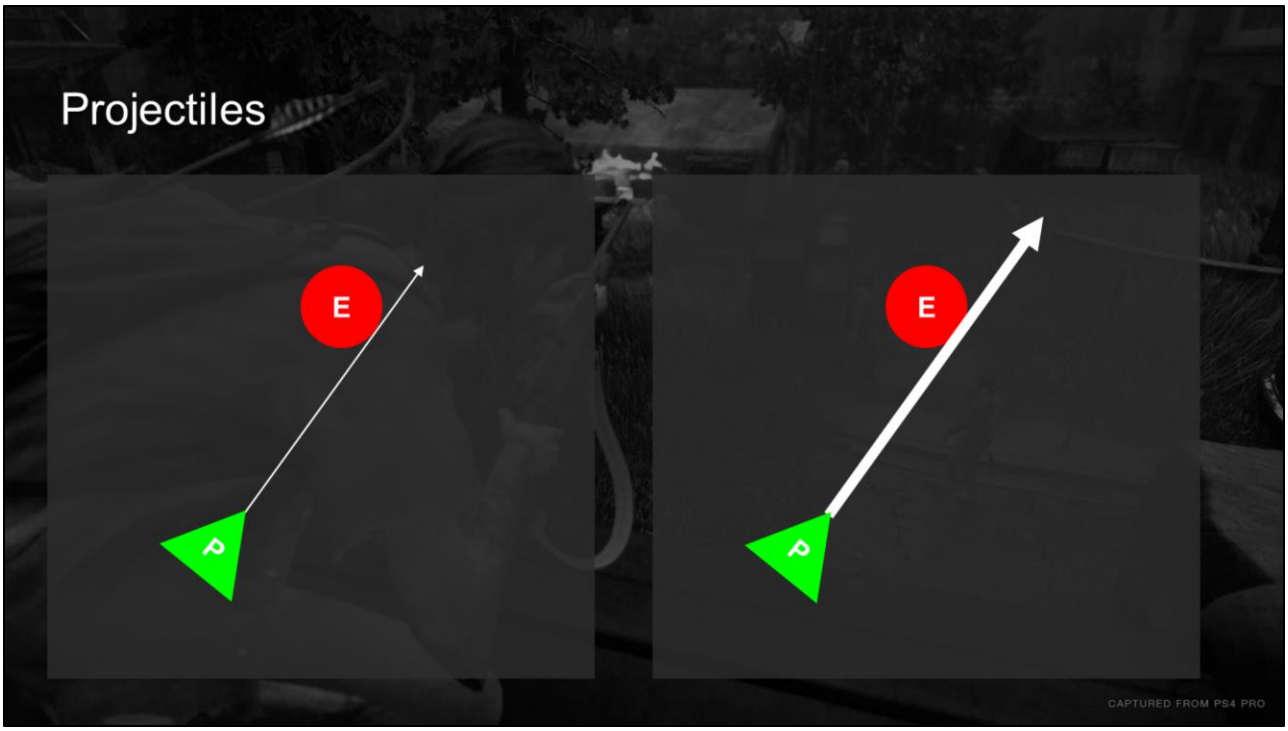
This was accomplished by adjusting gravity over time.

Instead of launching the projectile upward to get it to do what we wanted,

We use a more traditional vector forward through the camera reticle like we would for any other ranged weapon, (click) and disable gravity on it for the first few frames, blending gravity on over time.

You could also do this by implementing a drag system where the initial velocity is very high, and you manipulate the drag on it instead, but this was easy to implement at the time, and worked well for our needs.

So cool, we solved projectiles right? No other problems, we rock.



Wrong

We were getting lots of near miss cases with the bow and arrow, which was frustrating to players.

Somewhat due to frame rate and render delays (talk about later)

But the other largest issue we found was just the simplicity of the casting we were doing at the time. Turns out we were only doing a raycast for our probing which was not representative of the thickness of our arrows.

(click)

Solution: Support sphere casts instead of ray casts and bump of radius of probes slightly

Nice! Simple solutions to simple problems, right? Everything feels great.

Wrong again!

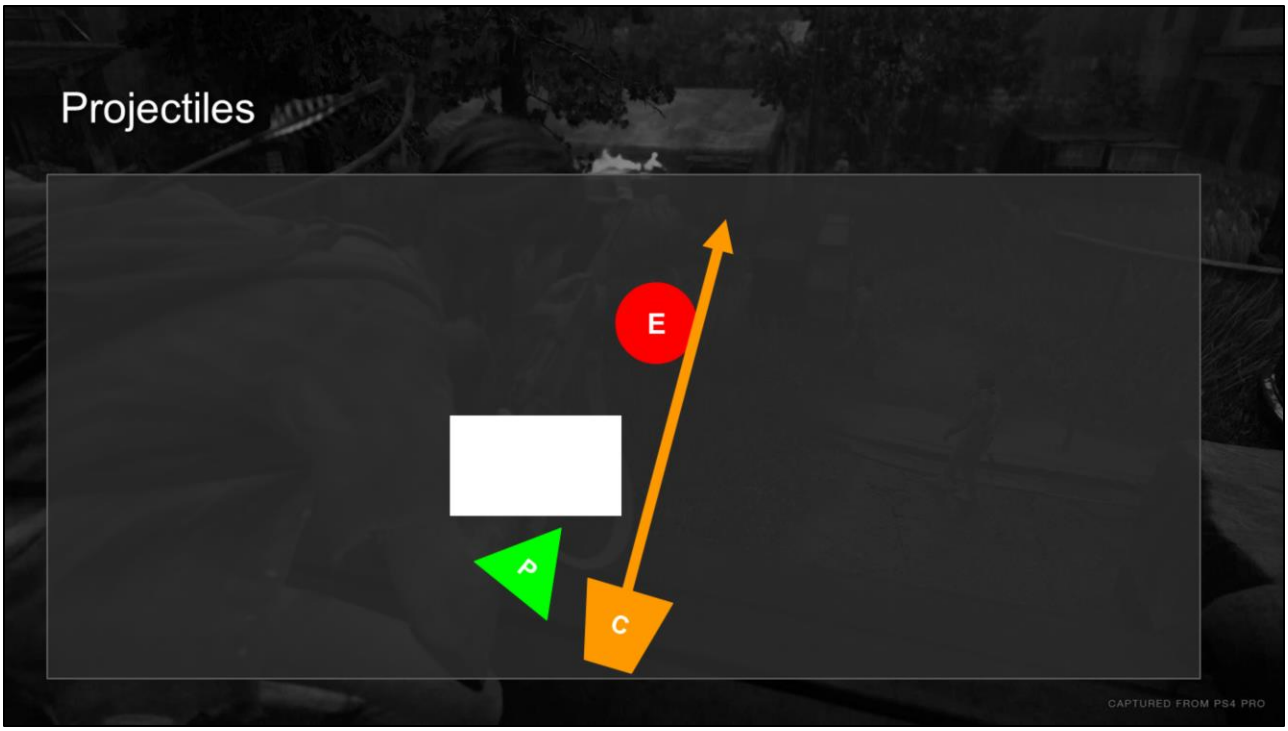
Let’s talk about the real issue with 3rd person shooting that is so well illustrated by Walaber’s debug in this tweet.

The decoupling of the camera from the player character.

Where does the projectile spawn from? Where does it go? What looks best? What feels best?

(Permission obtained from developer to include this in presentation)

Well, Let’s look at an instance of what we ended up shipping in the final game.



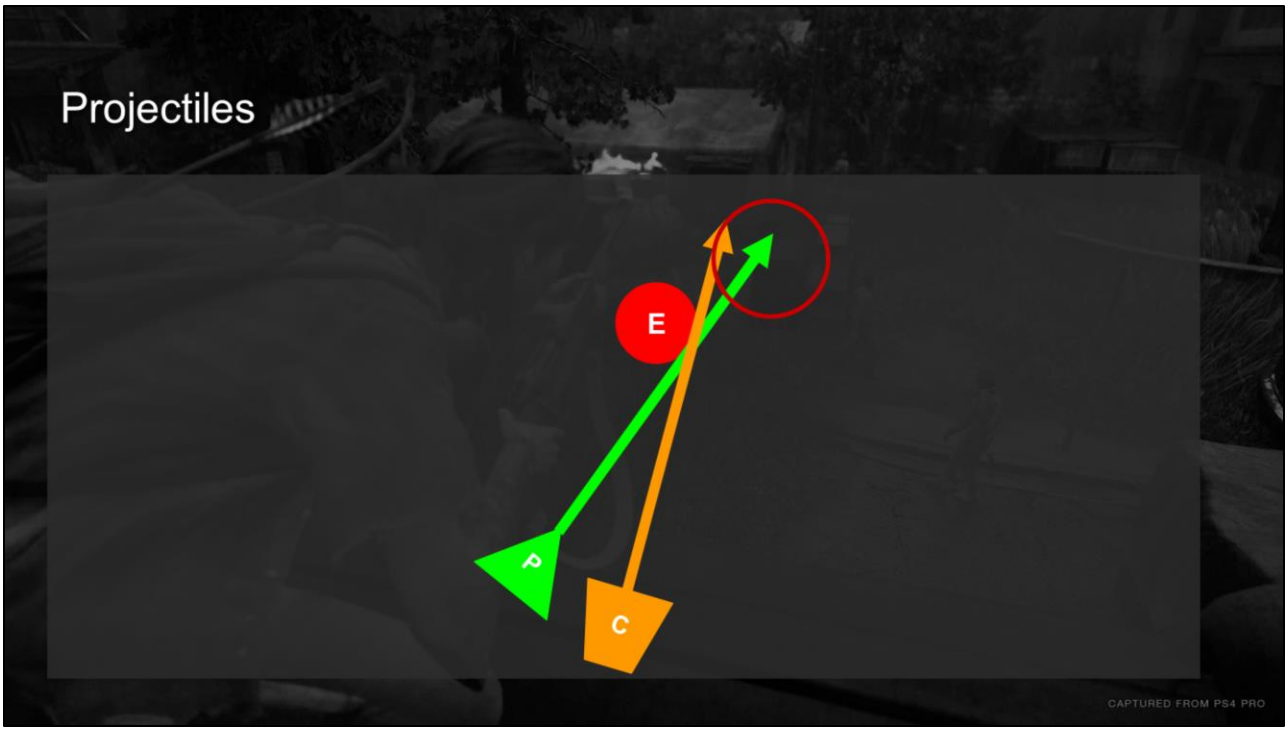
Ah right, so we must have chosen to shoot the projectile from the camera through the reticle and then just had the art just follow it right?

That’s an option, but…

(Next Slide)

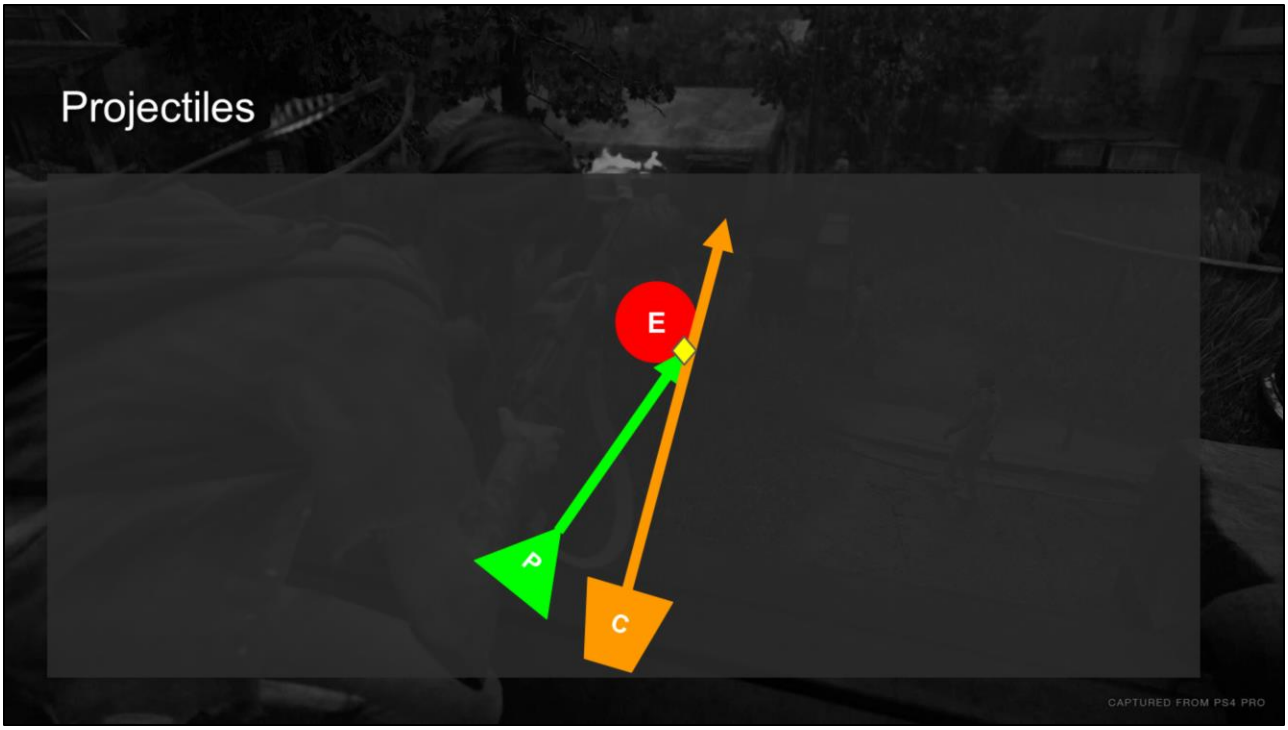
If you'll notice, the reticle isn't over the enemy here.

Which in the previous model should have missed right?



Okay, so we shoot two projectiles then? One from camera and one from the player character?

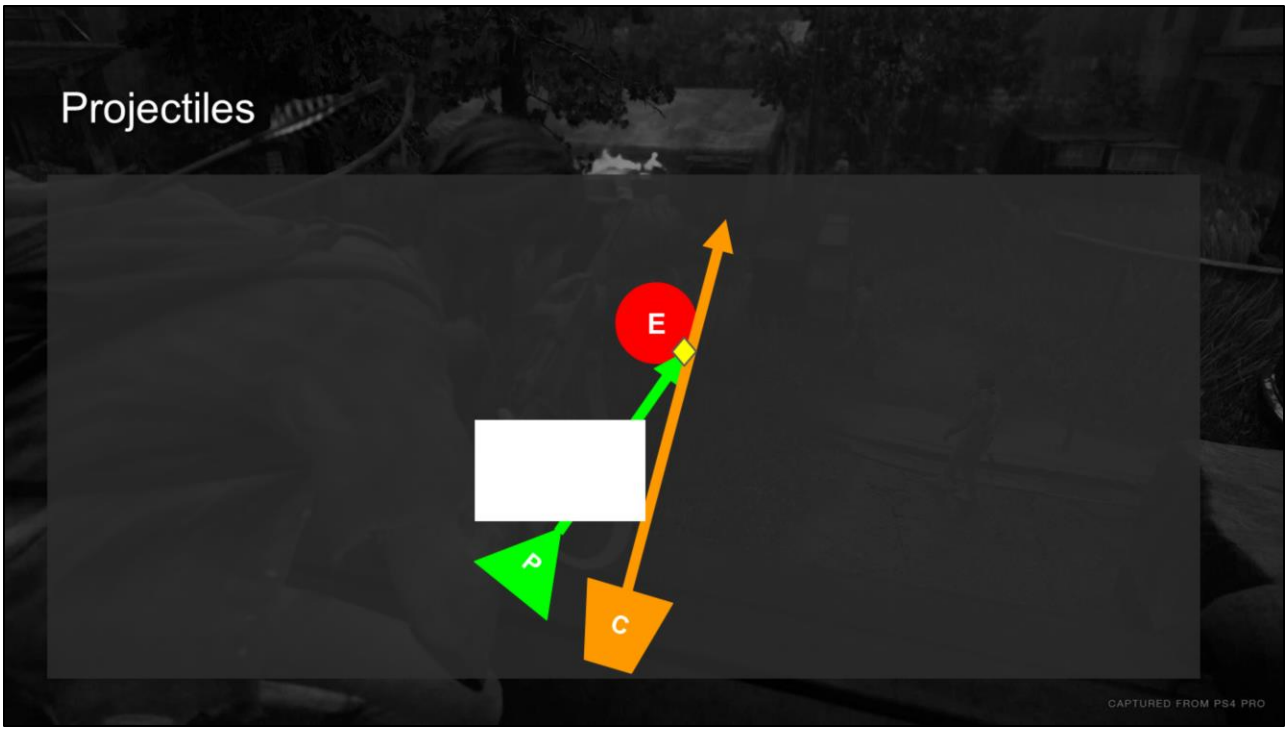
But where does that 2nd projectile go?



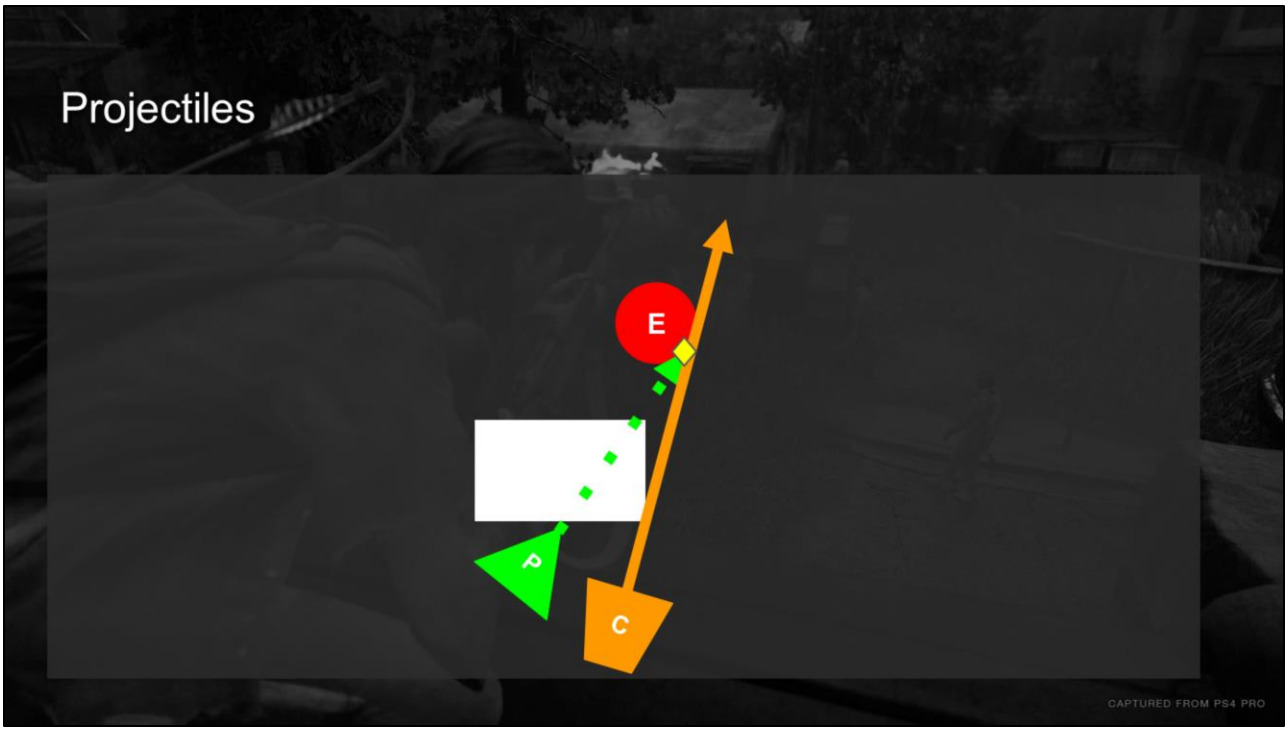
Well, let’s try using predicted collision point of the projectile from the camera (click) to back calculate the 2nd projectile’s trajectory.

(click)

Cool! That should cover it right?



Oh, but what if there is a wall blocking the way?



Collision Layers! We'll just make it so the 2nd projectile doesn't collide with our environment collision layer then.

Here is that exact shot at the start, but with debug enabled.

Let’s walk through it.

So our reticle is over the enemy, but there is a blocker in our way.

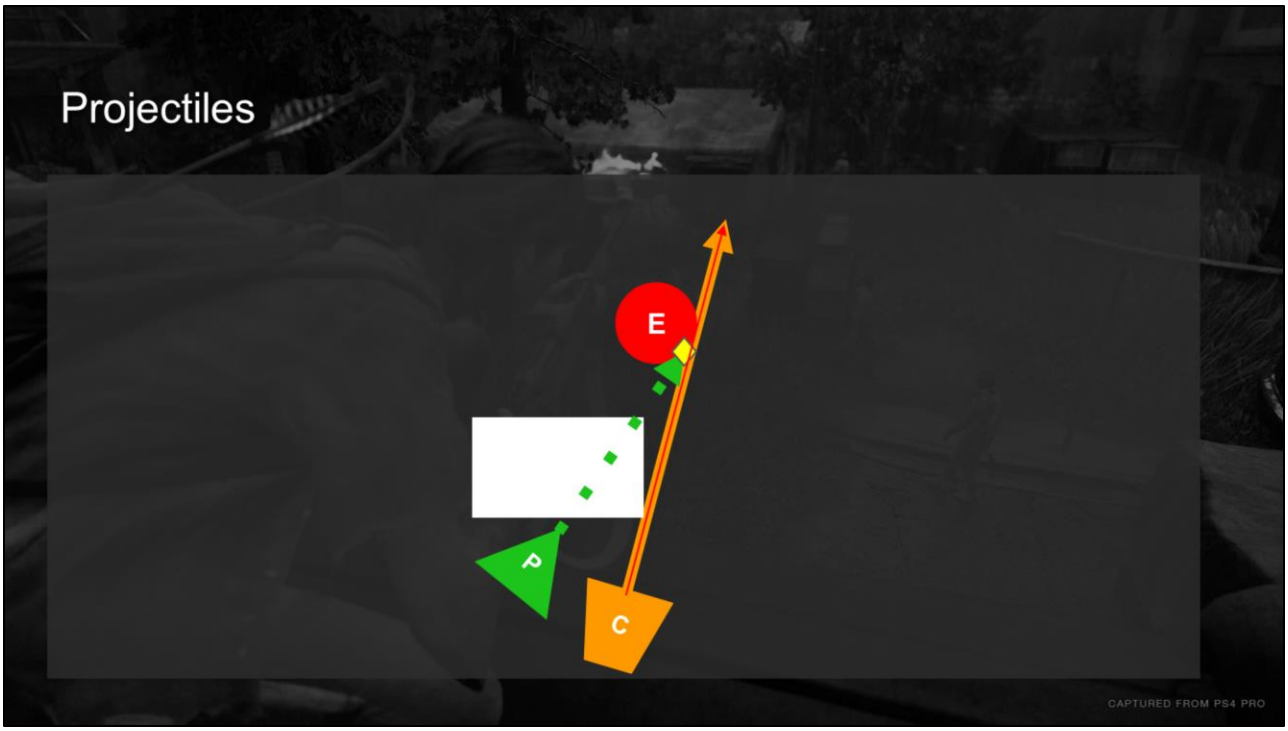
The trajectory of our player character projectile gets back calculated from the predicted impact location of the camera projectile.

As we shoot, the 2nd projectile collides with the wall, we throw that data away since it isn’t our target and let it pass through. This only happens for a frame or so, and most people never even notice it.

Then both projectiles follow their intended trajectories until the impact is made.

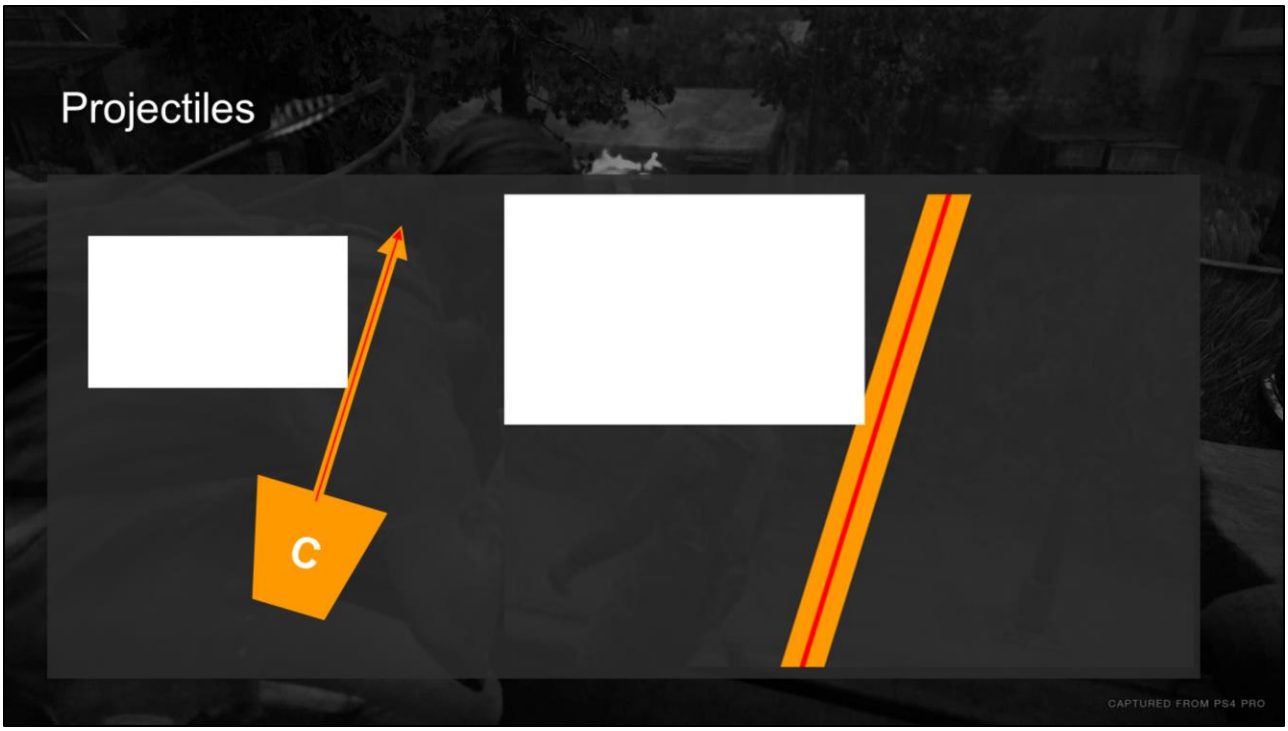
(Next Slide)

And our 2nd probe that only collides with enemies handles the case where an enemy is standing right in front of the player character, but the reticle technically isn't overlapping.

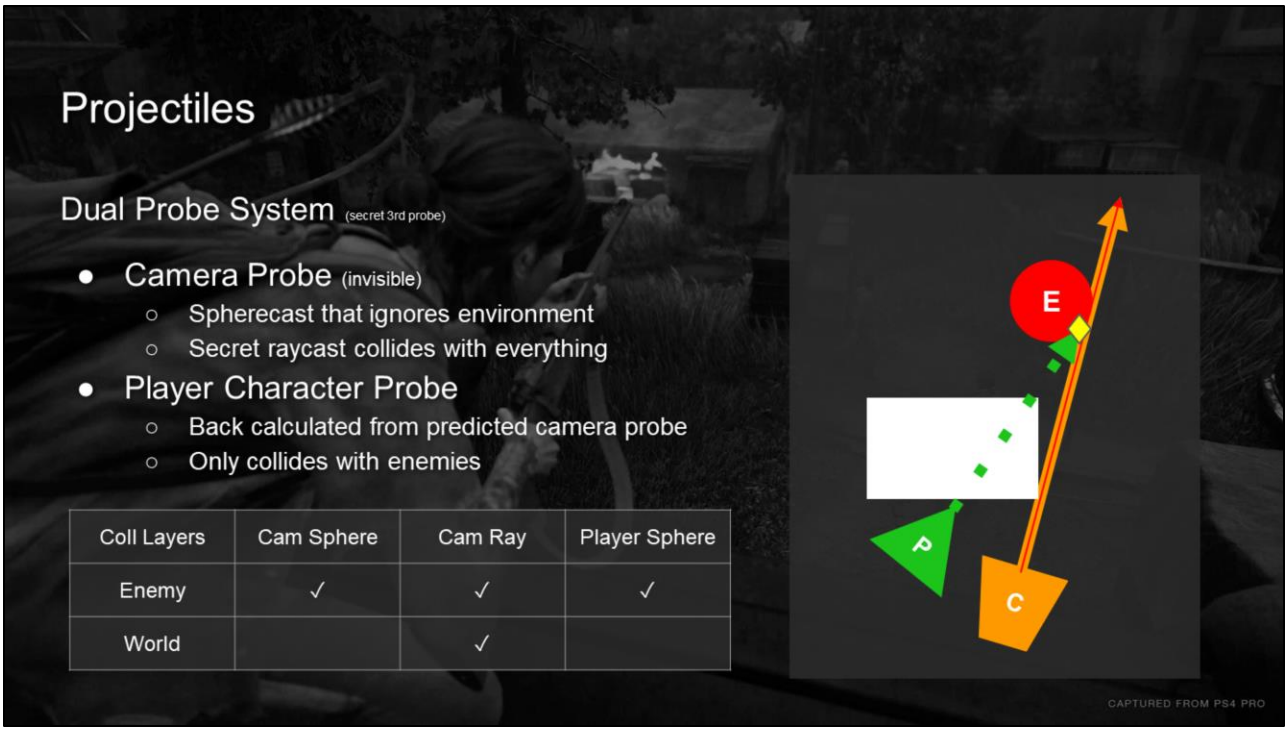


There is technically a hidden 3rd probe in here that is just a raycast.

If you paid attention earlier, you might have noticed that the camera probe is technical colliding with the wall due to its thickness.



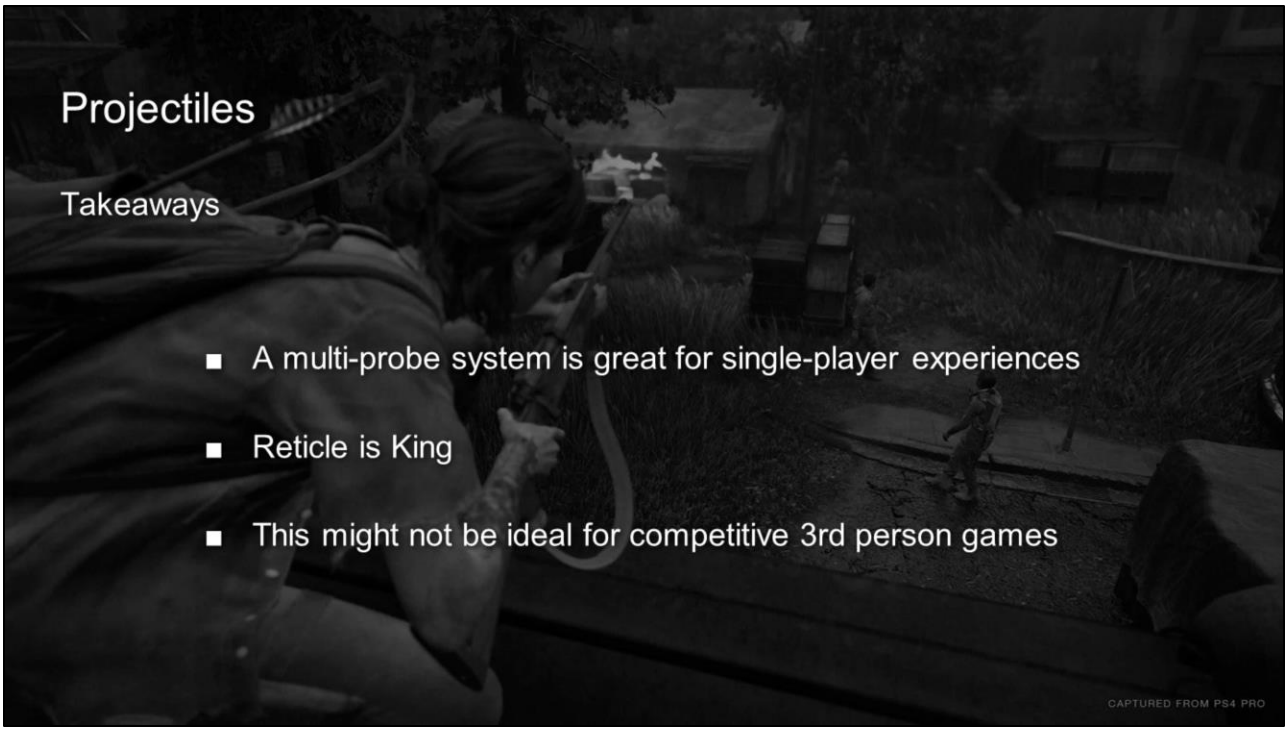
This third probe follows the camera probe exactly, but only checks for environment collision in order to further prevent accidental wall hits.



In Summary

Dual Probe System (secret 3rd probe)
(click)
Camera Probe (invisible)
Spherecast that ignores environment
(click)
Secret raycast collides with everything
(click)
Player Character Probe
Back calculated from predicted camera probe
Only collides with enemies

Special shoutouts to Erin Daly, Sandeep Shekar, and Kan Xu who worked with me on this issue and helped us come to this eventual solution.



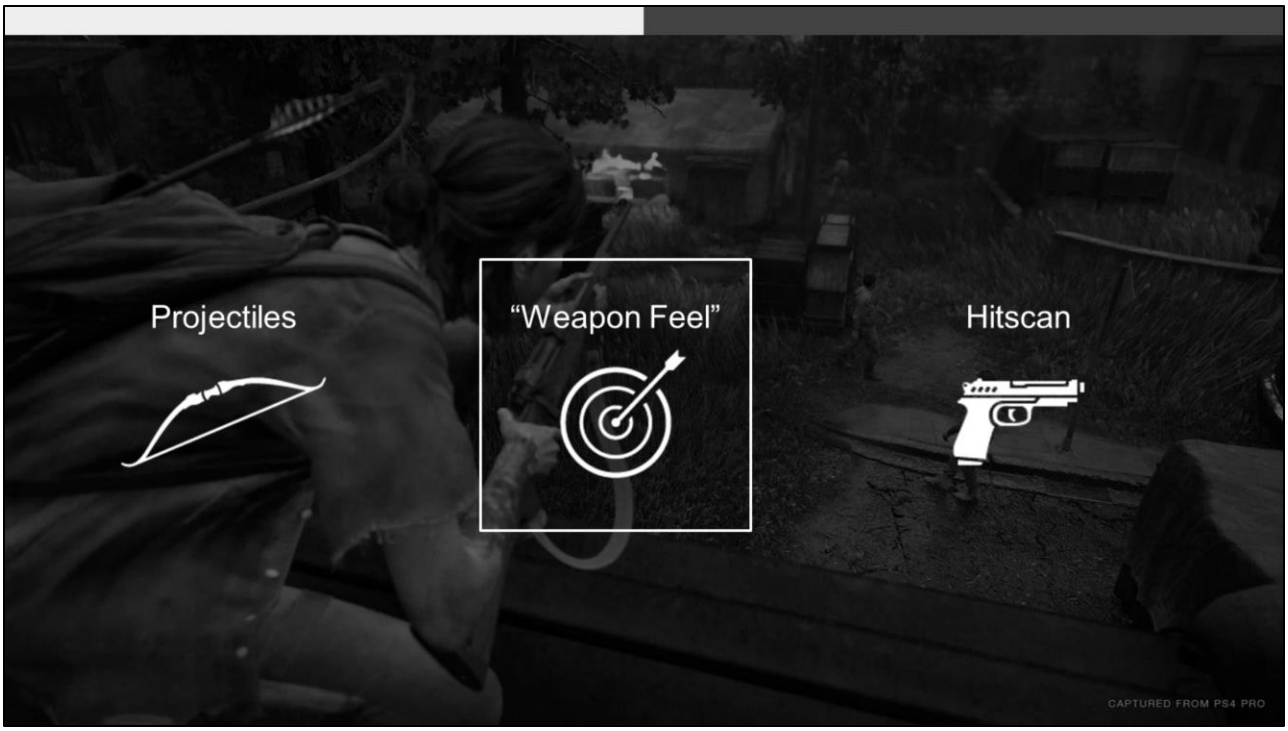
Takeaways

A multi-probe system is great for single-player experiences
- It's player favoring in so many ways

Reticle is King
- If you are using a reticle in your game, and it has shooting, it is expected to be reliable

This might not be ideal for competitive 3rd person games
- For the same reasons a multi-probe solution is so player favoring, it might not be ideal for a MP project.
- Imagine getting shot by another player that is fully hidden behind a corner.

Link to Forrest Smith's blog post on how to solve this in a Multiplayer Space
Adventures with Guns in a Third Person Shooter



Now let’s dig into this “Weapon Feel” topic

What Is Weapon Feel?

(Hold for response from Audience)

(Click)
VFX? Sound? Firing Rate?
(Click)
Camera Shake, Kick, Firing Patterns?
(Click)
Controller Vibration?
(Click)
PS5 Trigger Haptics!?

Yes, it is all of those things, but…

I would argue

It is almost entirely hit reactions.

When hitting another animate object, especially a human, our brains just expect them to move in certain ways.

Humans are insanely good at recognizing patterns, and that means we are also insanely good at noticing when they are not there.

Next Slide

Showcasing one of our talented technical animator Jason Lei’s reel;

You can see we really wanted to push the fidelity of hit reactions from the studios previous games.

Anywhere from shooting a character in the leg being honored with an appropriate reaction to shooting a human in the neck receiving a unique death animation, even though under the hood gameplay wise it is still just a “headshot”.

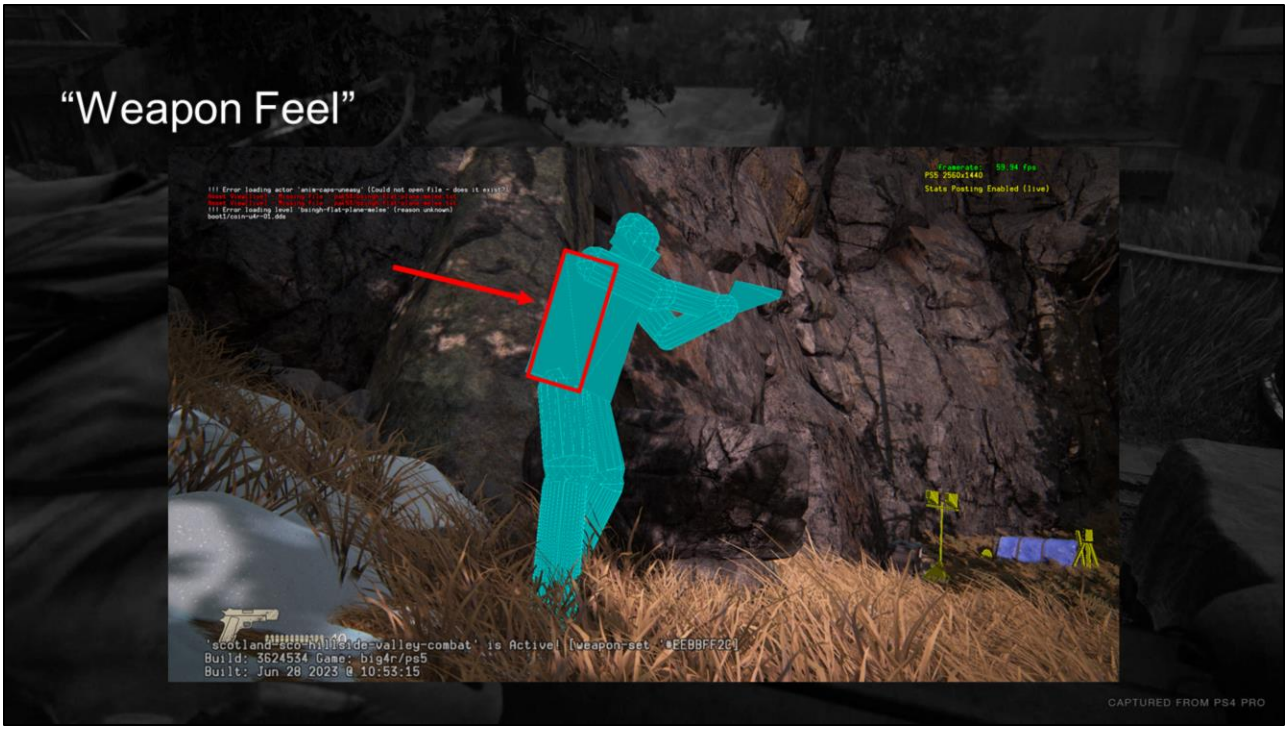
But in order to do all this, we really needed some other things to be improved first.

Most notably, our hit collision.

Here you can see the hit collision boxes for Uncharted 4 npcs.

If you look closely you, might catch a few potential problem areas.

(Next Slide)



See it yet?

(Click)

Right here.

(Click)

Not sure how acutely everyone has studied the human torso, but while it might be somewhat box like in shape…

(Next Slide)

It is certainly not as rigid.

We bend and move our torsos about in all kinds of ways…

(Next Slide)

That a rigid box will simply not conform to.

(Next Slide)

(Hop back and forth between previous slide and this slide as needed to illustrate point)

In order to support our goals of fidelity, it also meant that we needed some better support for hit detection and collision.

After working with Jaros Sinecky our resident collision and physics guru and the character department we worked out a better pipeline for creating character collision that allowed us to more easily iterate on our hit detection.

To resolve the torso issues, we ended up going with multiple horizontal capsules attached at various spine joints up and down the character.

(Next Slide)

This resolved many of the unusual hits and misses caused by characters being in more organic poses.

This improved hit detection also helped us in other ways.

Issue
When first taking on ranged combat systems I yearned for a larger selection of ways for the player to express themselves.

I also often found it was easy to get overwhelmed by enemies and was wanting for more crowd control options.

Previous Player Expression Options
Body Shot (Regular Damage)
Head Shot (More Damage)

I expressed this want to Matthew Gallant the current Designer on Human Enemies and after some back and forth with animation on what level of complexity we could afford, we were able to add a leg stun state.

Solution
Add leg shot stun state
Instant stun state, less damage

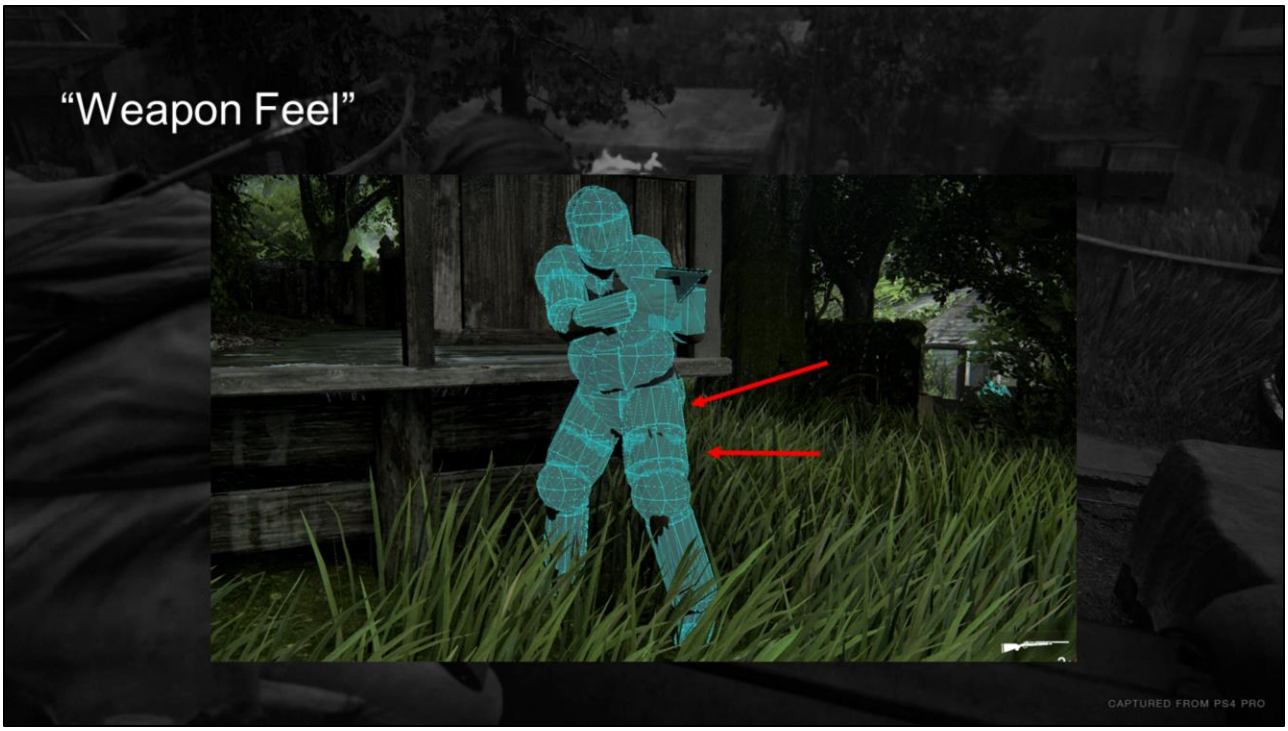
Which opened you up to doing a few things.

Just like a bottle or brick:

You could use it as an opener to move in to grab an enemy as a meatshield while you deal with other threats in the area.

Or immediately trigger a strike attack.

Additionally, when hiding in prone crawl spaces, you could use the leg stun state as a way of bringing an enemy’s head closer to the ground, opening them up for a fatal shot.



Fun anecdote

Early in playtesting with the leg shot stun mechanic, we found that players would often aim at the knee, and occasionally miss the leg stun because it would hit the thigh capsule instead of the knee.

So we ended up splitting the thigh into 2 collisions.

(click)

One that was included in the core of the body

(click)

And one that was included with the lower leg, this made the mechanic a bit more player favoring, and prevented a lot of “near miss” cases.

Not to mention it was much easier to do with the improved character collision pipeline!

Higher graphic fidelity creates an expectation of higher fidelity in gameplay.

This ties into the Mental Model vs Design Model concept - Design of Everyday Things - Don Norman "Door"

The larger the gap is between the player's perception of the world, and the actual design of the world, the more jarring it will be when the world doesn't interact like we expect it.

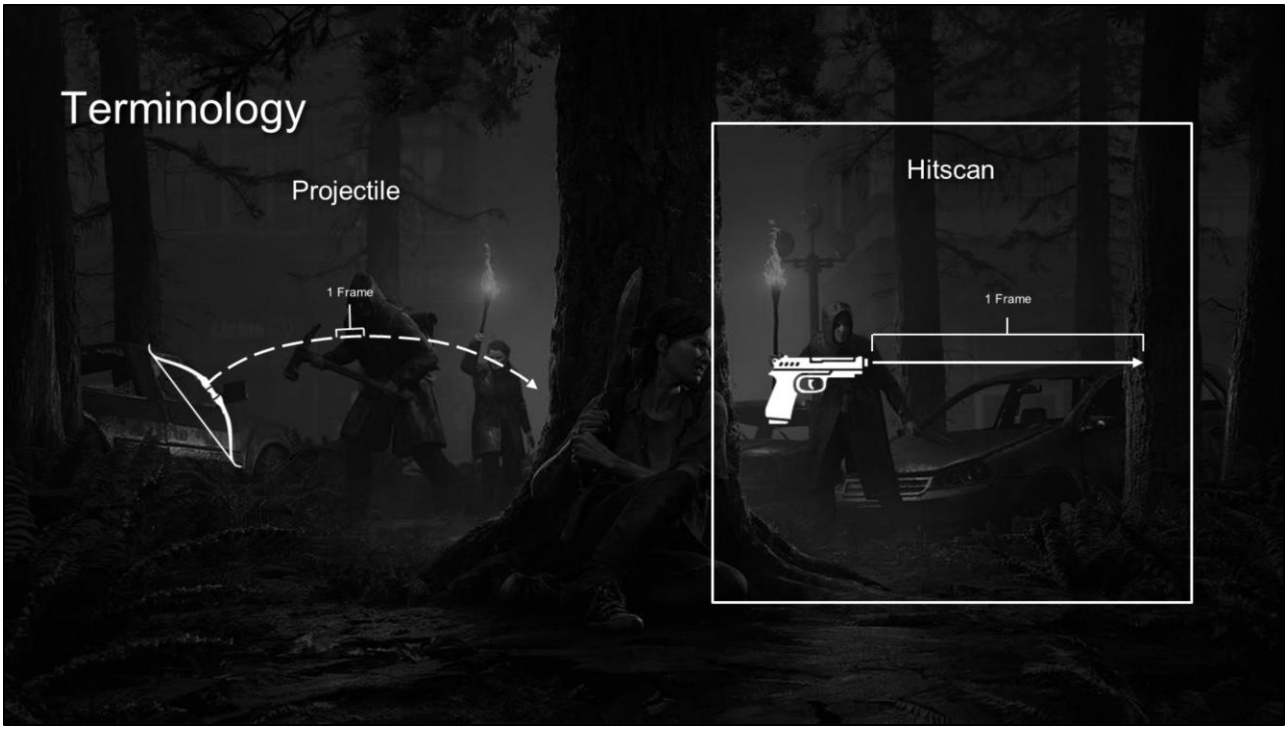
Assume nothing. Question everything.

Processes always have room for improvement, even at the storied studios like Naughty Dog. Don't assume that the current process exists because it is perfect. For us that was character collision, but this holds true for any pipeline or process.

Start with a good base before focusing on detail.

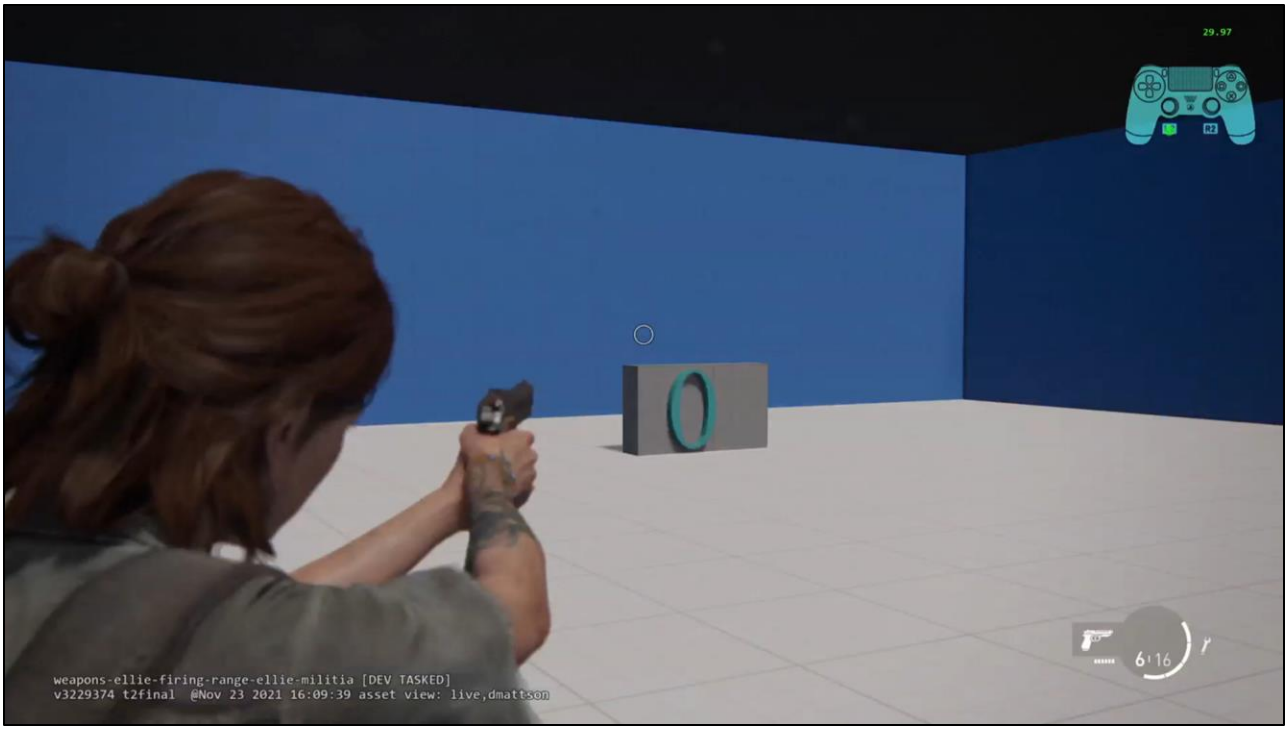
You can add all of the the ancillary feedback you want. FX, Sound, Controller Haptics, but if you don't start with a good base of animation and hit detection. It can add noise making it harder to realize the larger issues at hand.

Now last, but not least. Hitscan.



Which if you remember from the beginning of our presentation is the single frame probe.

Hitscan - Weapon probes for 1 frame, everything else that lingers is just feedback



Our base pistols in TLOU2 are hitscan weapons.

Let’s take a look at a moment of gameplay.

Let’s watch that one more time actually.

(click)

What you probably don’t realize in this video, is that the frame in which the engine thinks the player fired their weapon is

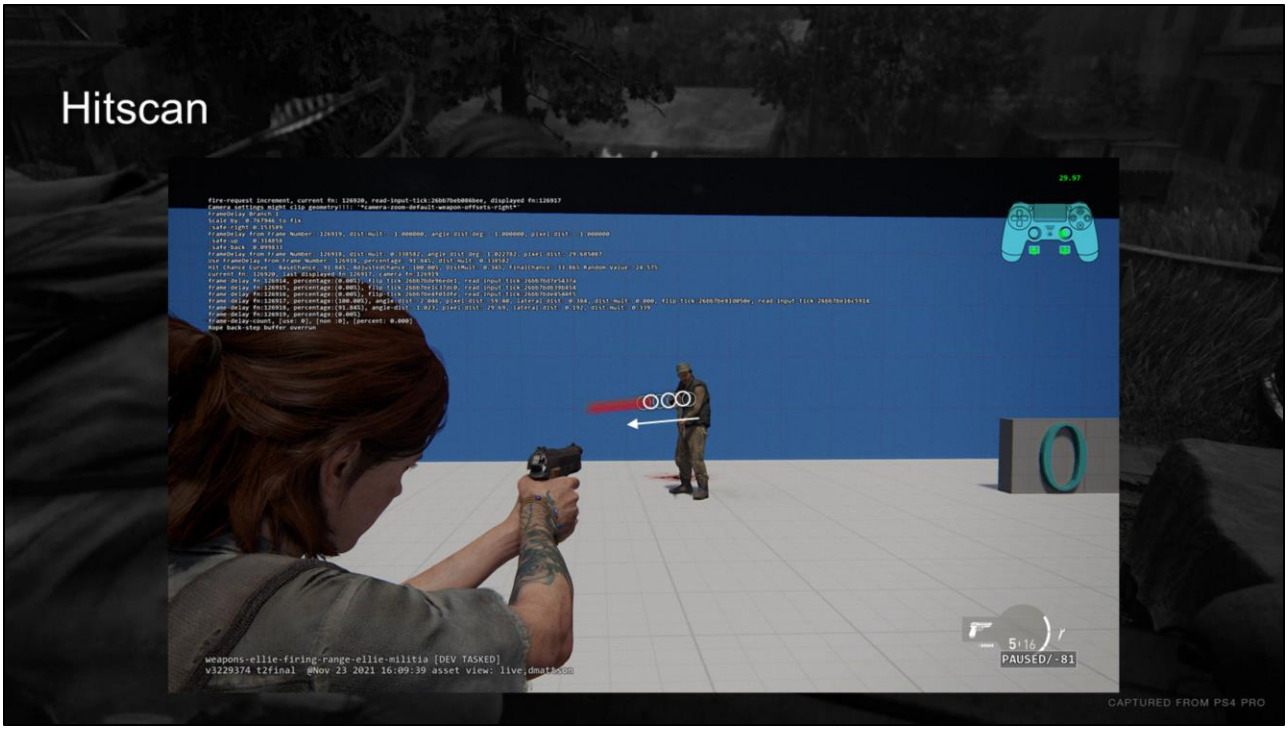
(click)

This frame.

But we shot the enemy right?

Or at least it looks like we did.

(click)

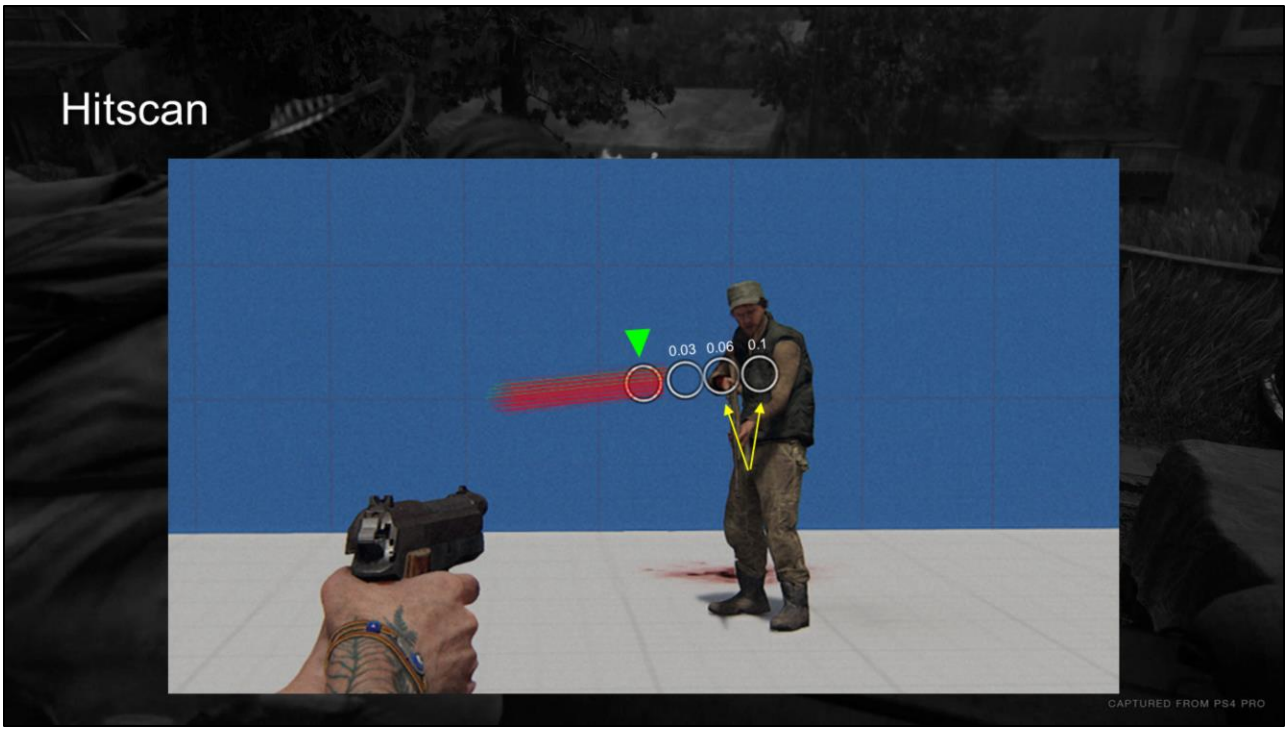


Here is an image composite of the 4 frames before the engine renders the hit sweeping form right to left overlayed on top of each other. All in all this represents about 0.1 seconds of time at 30fps. This may not sound like much, but as many of you may know in an action game we measure time in ms not seconds.

I know everyone's first thought, just make the game run at 60fps right?

Now remember that our project and studio has a goal of pushing graphical boundaries.
One of the ways the studio is able to do that is utilizing a lower frame rate to allow more time between each frame for the render.
This means that our target on PS4 for TLOU2 was 30 frames per second.

But just for reference, here's a draw over of what this could look like at 60 fps for reference. Way more opportunities to query from.



Our engine has a double frame or triple frame pipeline depending on its target fps.

The exact ins and outs aren't my specialty, but I know it has something to do with giving more time for the engine to render a frame well.

What I do understand is how it affects gameplay, and the result is that what you see and react to in game can be up to 3 frames delayed from your current input at 30fps. (excluding any outside factors like input lag or tv refresh rates).

Now what does that mean to the end user?

(click)

It means that if this frame, is when we render the shot.

(click)

That the player pushed their input while viewing one of these 2 frames.

(click)

Which remember at 30fps is somewhere between 0.06 and 0.1 seconds in the past.

(pause)

Put simply, it means that it is easier to miss shots and feel like the game robbed you, because in a way it kind of did!

(Next Slide)

In order to compensate for these near miss shots, we came up with a solution that would store the history of the player's aim from the past.

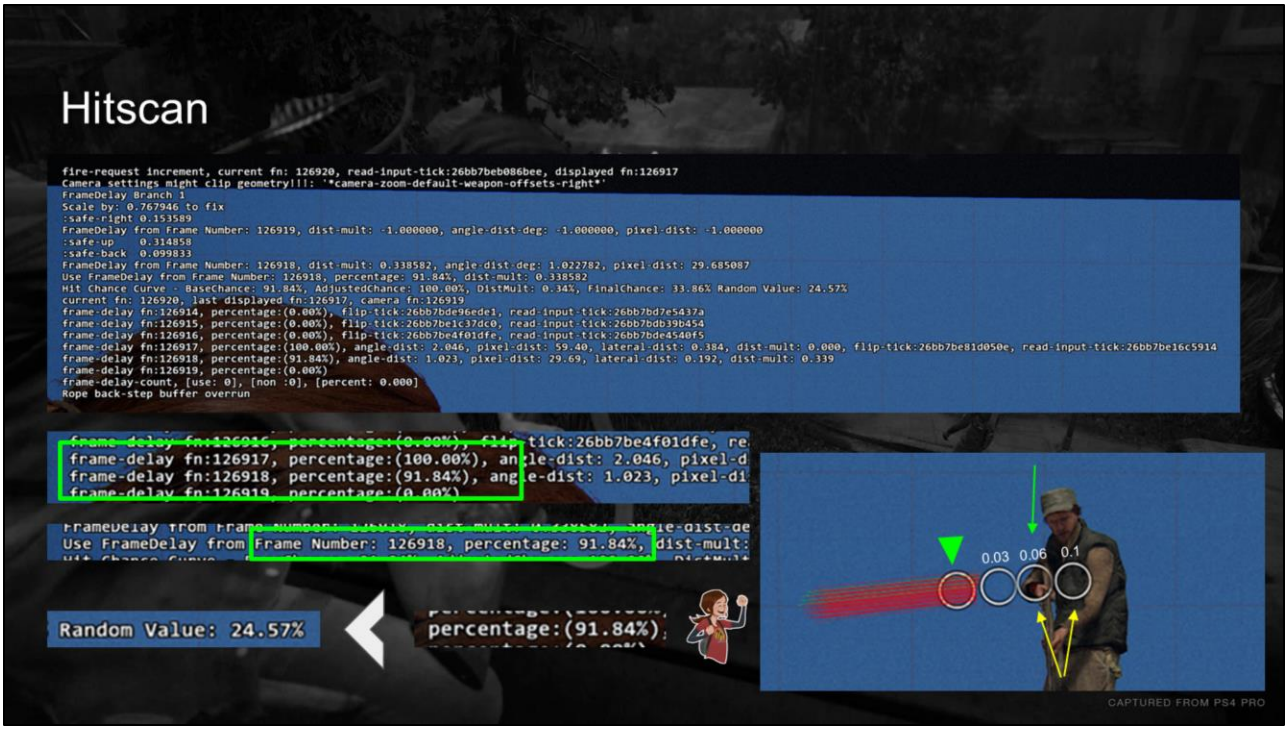
Our solution to this was to send out a number of casts every frame, and store the percentage of the reticle overlapping in a buffer for the past few frames.

(click)

This meant we could look into the past, when we think the player had pulled the trigger and determine whether or not they deserved the hit.

Note on this: We only ever do anything with this information if the player misses a shot, and within the past few frames had overlapped a valid target.

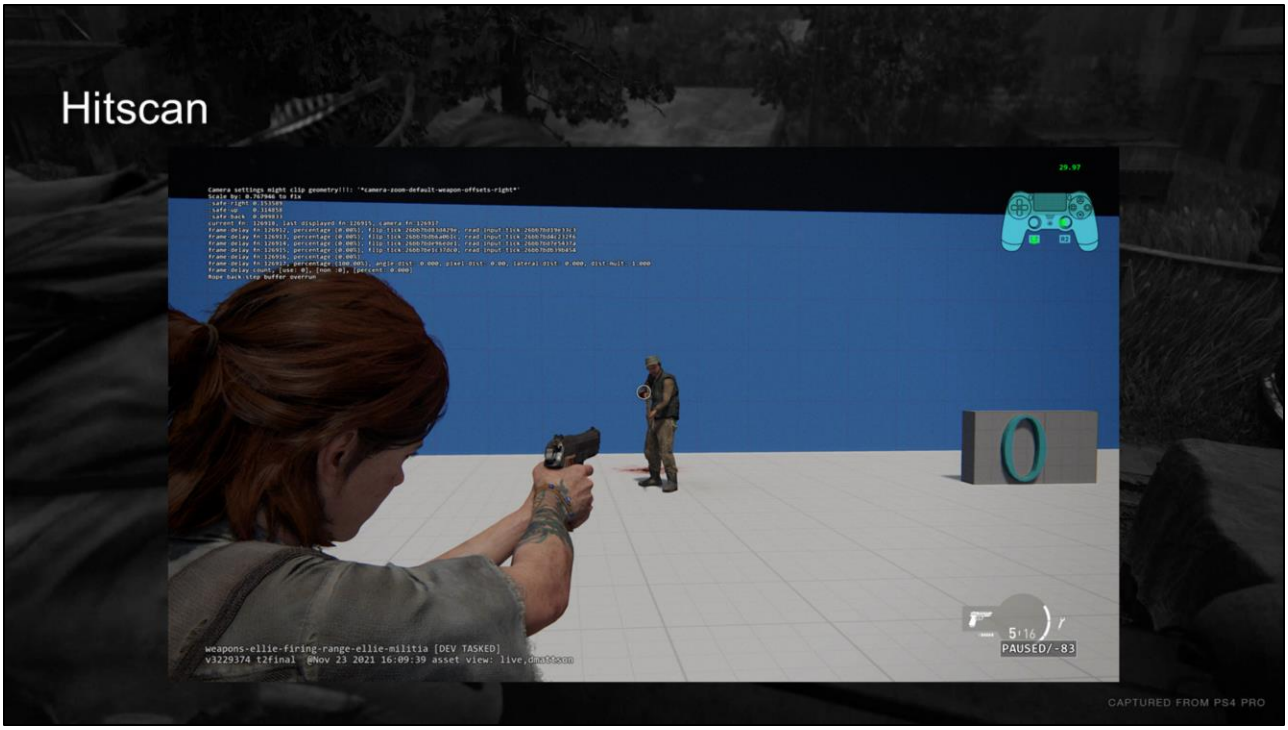
(Next Slide)



In our example case
(click)
It looks like we are overlapping the enemy in both frames that we could have pulled the trigger on.
(click)
The frame delay gets selected, and it is our 91.84% chance frame.
(click)
We do a random dice roll, and as long as the value is less than our 91.84% value, we give the player the hit!
(click)
(Next Slide)

Don’t worry, there’s no more wrenches that could be added to this complex problem. Right?

WRONG AGAIN!



Let’s talk about this frame.

Yeah this one, notice anything about it?

Perhaps you have heard of…

(Next Slide)

The XCOM problem
(Next Slide)

Oh no, does this look like slightly less than 100% of the reticle!?

Why yes it does.

Meaning that even with an extremely high chance of hitting, you could in fact still miss this shot.

But I think anyone here could rightly argue that this looks like a guaranteed shot. Especially In the heat of battle.

(Next Slide)

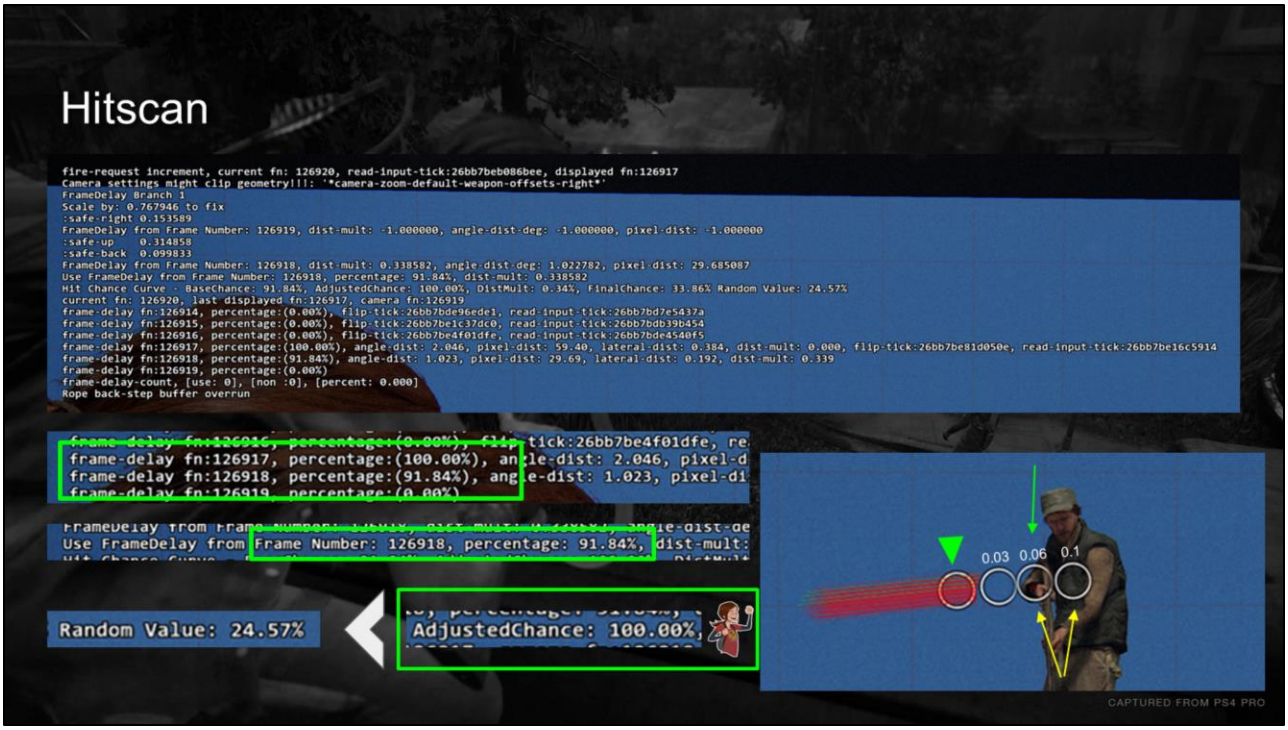
Solution?

Just make 95% == 100%!

We ended up adding what we referred to as a hit chance curve to the engine.

Where we would take the stored percentage value and recalculate it based on a curve that we thought felt best in game.

(Next Slide)



So when that Frame Number returned 91.84%, run it through the curve and return an Adjusted Chance.

(click)

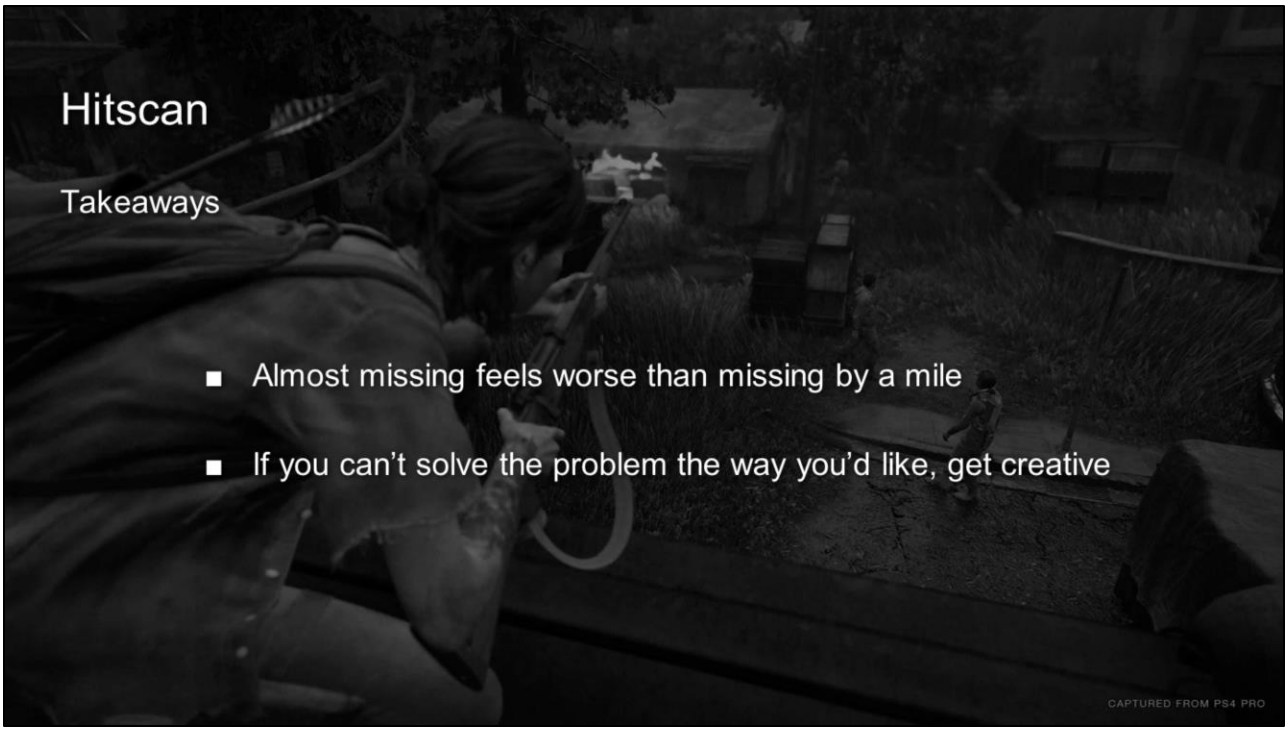
Of 100%!

Giving the player who just narrowly missed it

(Next Slide)

A Second Shot

(Next Slide)



Almost missing feels worse than missing by a mile

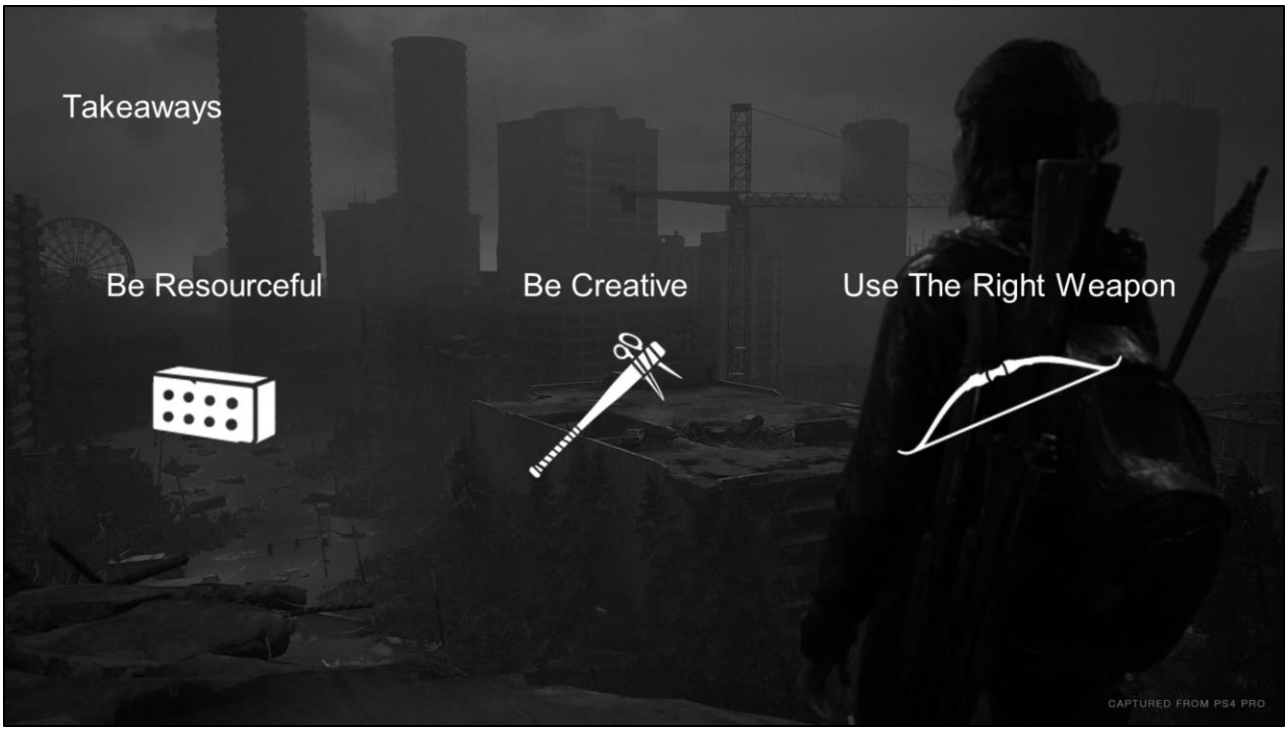
These are the kinds of low frequency, high impact problems that cause players to stop playing your game. Evaluate appropriately for your project, but while investing time on solving these kinds of problem might not look important on a spreadsheet, it makes a world of difference to the overall experience.

If you can’t solve the problem the way you’d like, get creative

Some of my favorite solutions in games are workaround solutions to the limitations of the project or engine. They’re effective and they work. I’ll never tire of “Design Hacks”

(seriously, please feel free to send me your hacks/workarounds)

While there were tons of other improvements made between the games, these were the 3 I thought had the largest impact, most of which were entirely invisible to the player.

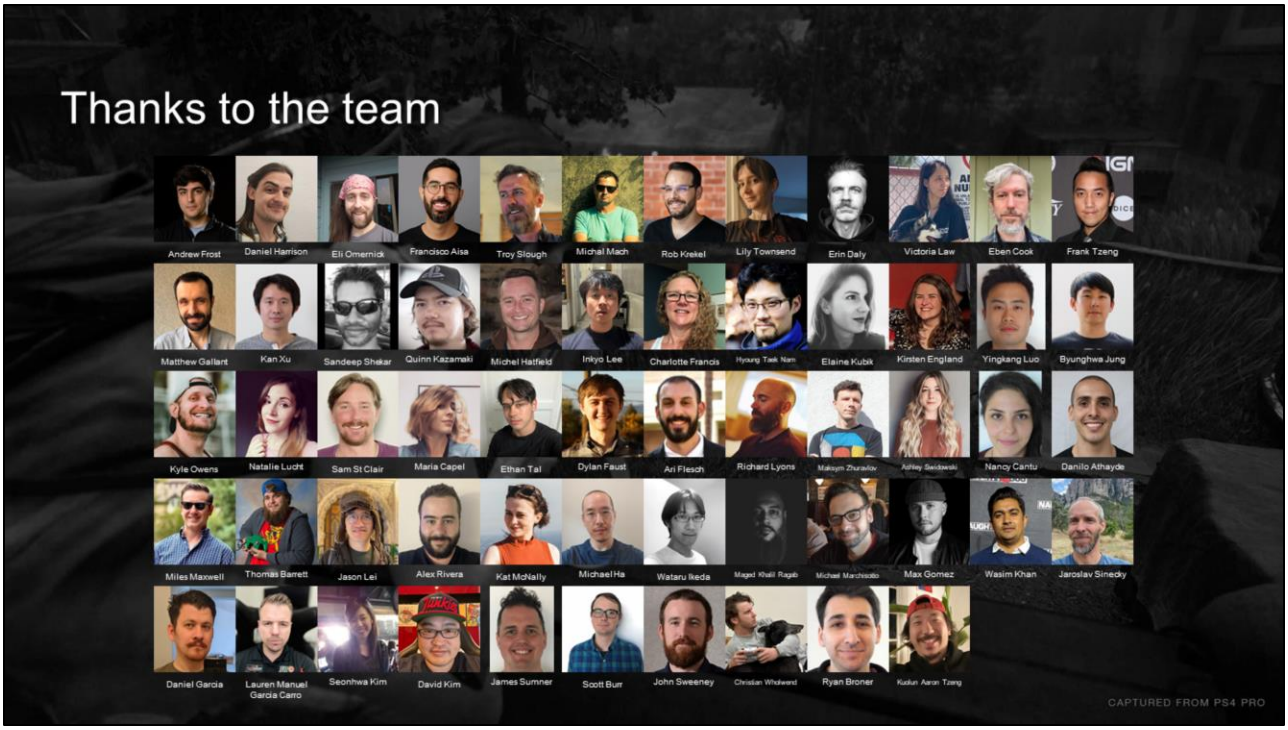


Use other games as a reference.
If someone else has solved this problem, try it out, you can always try a different solution later.
Designers are more like chefs, than inventors. Creating unique dishes from the same base ingredients. - Ben Brode - GDC Marvel Snap talk.
AKA Don’t reinvent the wheel

Use creative solutions for difficult problems
The Second Shot mechanic doesn’t solve the core issue of input lag within our render frames at lower framerates, but it was a cheap enough bandaid to try, and it gave us the tools we needed to create the end user experience we were aiming for.

Make the right decisions for your project.
Make sure you understand your project goals when making decisions.
Examples:
High Graphic Fidelity
Higher graphic fidelity creates an expectation for higher fidelity in gameplay. In order to meet player expectations and not break the 4th wall.

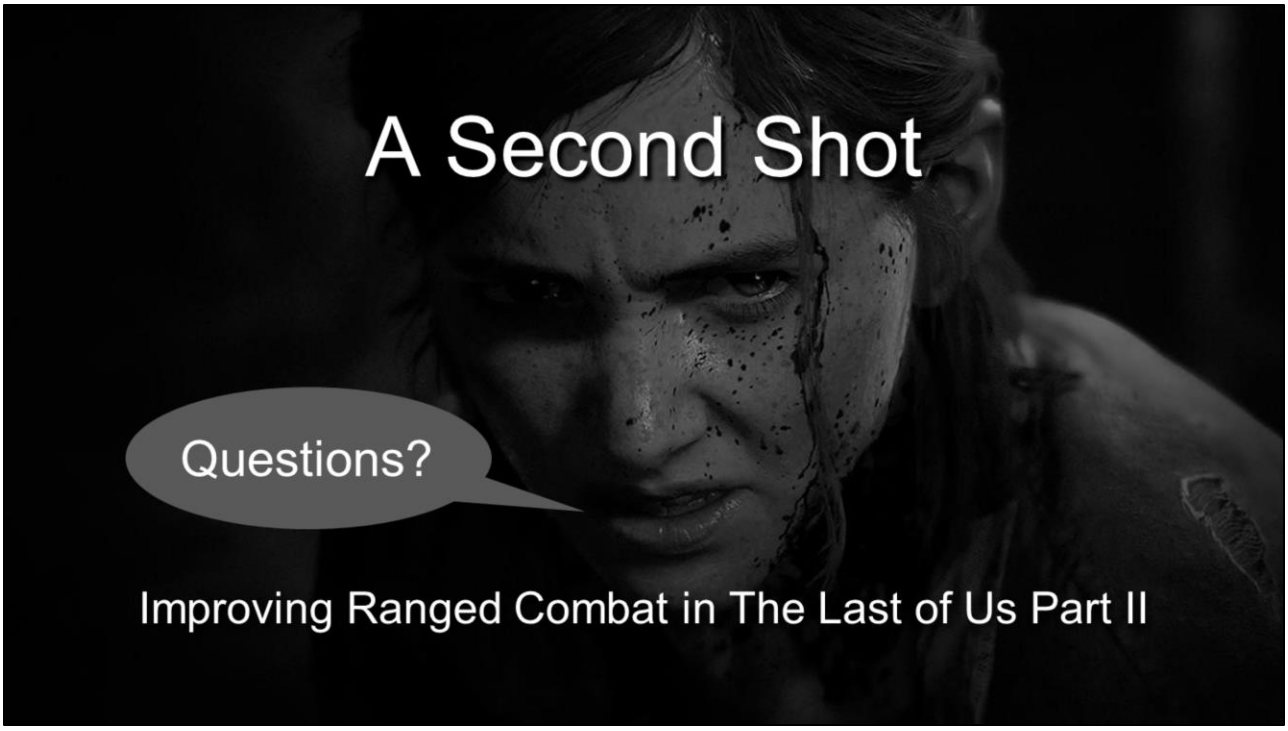
Taking the player on a journey through the narrative
Gameplay should not prevent the player from experiencing that narrative



Who made this happen?

Turns out, it isn't just the person on stage in front of you. A lot of people from a large variety of disciplines contributed to improving ranged combat!

I certainly could not have done anything without these lovely folks.



Thanks again for coming to the talk.

Don’t forget to fill out the feedback forms that are emailed to you afterward.

(Click)

Questions?