FIGHTING LATENCY ON CALL OF DUTY BLACK OPS III

BEN GOYETTE

ENGINEER AT ACTIVISION

Talk Description:
Latency in multiplayer games is hard to fight because it is hard to measure. Classical profiling techniques involve looking at network and CPU captures and they reveal the obvious offenders. But when everything there looks fine and the players are still complaining that it doesn't "feel" right, where do you go next?
For Black Ops III high-speed cameras and instrumented gamepads to see what was really happening frame by frame were used. This talk will discuss how the team analyzed the latency of shooting, jumping, changing stance, strafing and damage feedback. The talk will also look at the local input lag, the line-of-sight advantage of client-side prediction, the perceived time-to-kill and the effectiveness of lag compensation techniques.
Benjamin will present what was learned using this new measurement technique and show where an unsuspected amount of latency was found.

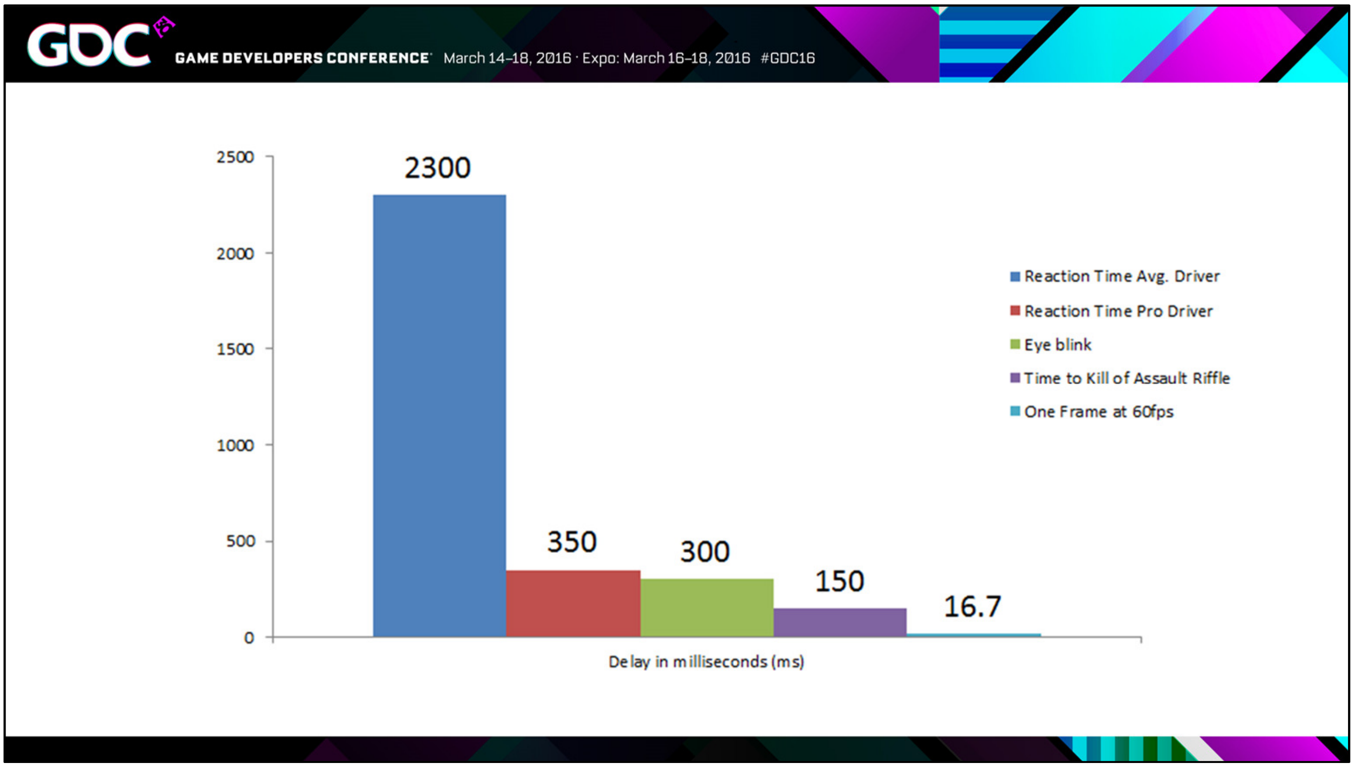
Takeaway:
"The camera don't lie". Attendees will learn the benefits of measuring latency using an high-speed camera. They'll learn how to create their own latency testing setup to verify their code's true performance. Because the camera sees what the players will see. That other profiling tool might just be lying.

Intended Audience:
For network programmers looking for new ways to measure latency and insure that their game is as fast as they think it is. No programming knowledge required.

Call of Duty is a fast paced multiplayer first person shooter.

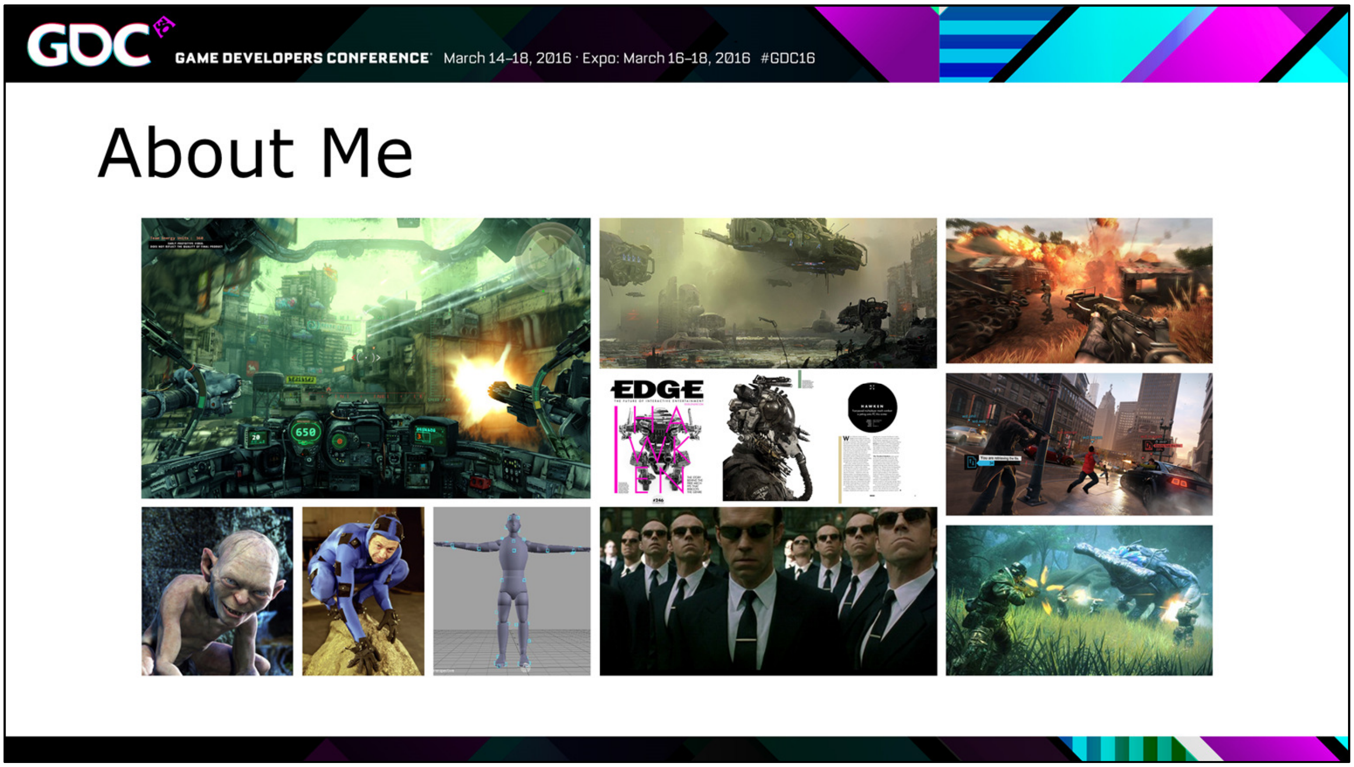
It feels amazing when everything is smooth and responsive.

That’s why we don’t want to introduce any unnecessary frames of latency.



“Milliseconds (ms)” is the time unit used to express latency.

Reaction time of an average car driver: ~2.3s = 2300ms
Reaction time of pro drivers: ~350ms
Time to blink your eyes: ~300ms
Death by Assault Riffle in COD: 150ms
One frame of latency at 60fps: 16.7 ms

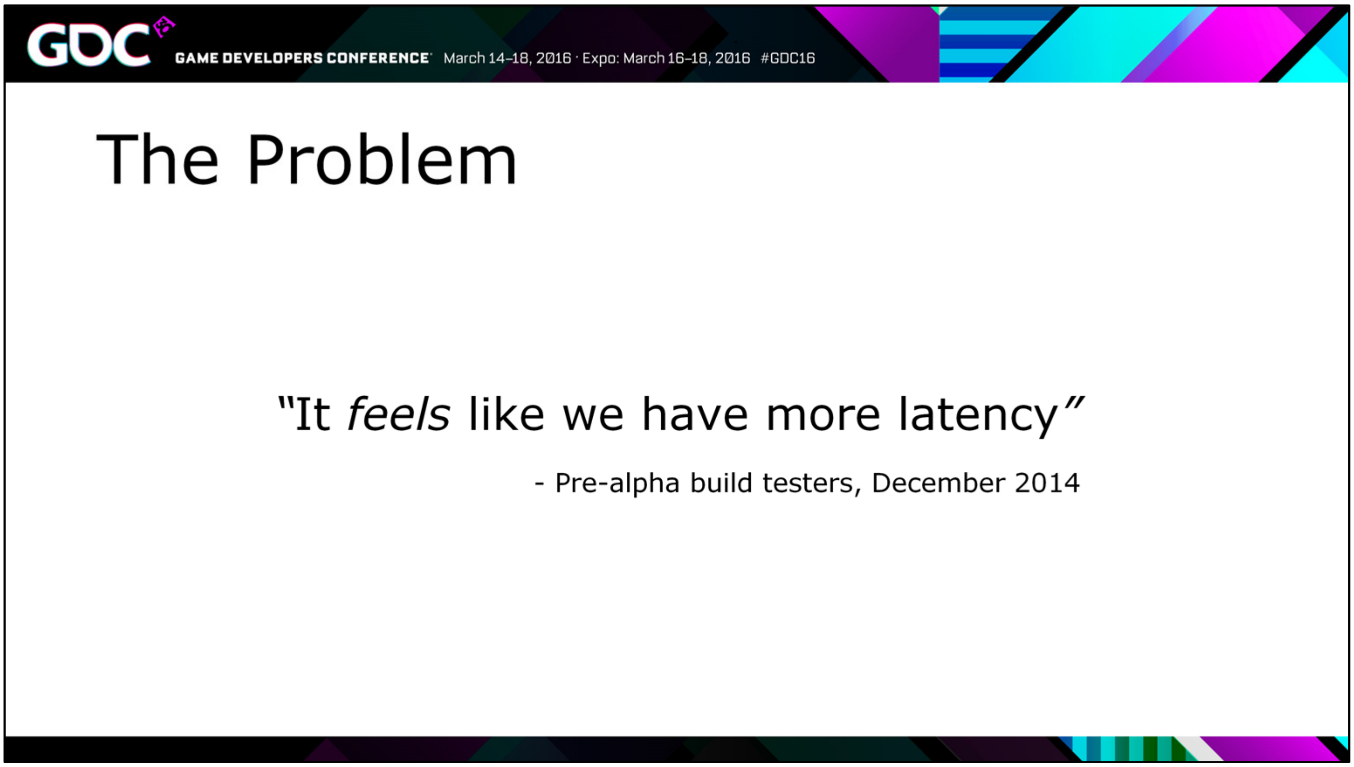


About Me

Ben has been an engineer in the industry for over 13 years. He started as a programmer on Kaydara FiLMBOX that became Autodesk MotionBuilder, an industry standard tool for motion-capture and real-time animation. He worked for Ubisoft Montreal on the development of the Dunia and Disrupt game engine pipelines. He built the foundation of the Storm Network Engine used for synchronizing the company's open-world multiplayer games. He was the Lead Engineer on the indie mech shooter HAWKEN and is now at Activision using all his real-time engine experience to fight latency in the eSports shooter Call of Duty.

About Activision Central Studios: A group that supports studios owned by Activision.

https://www.linkedin.com/in/benjamingoyette



The Problem

"It *feels* like we have more latency"

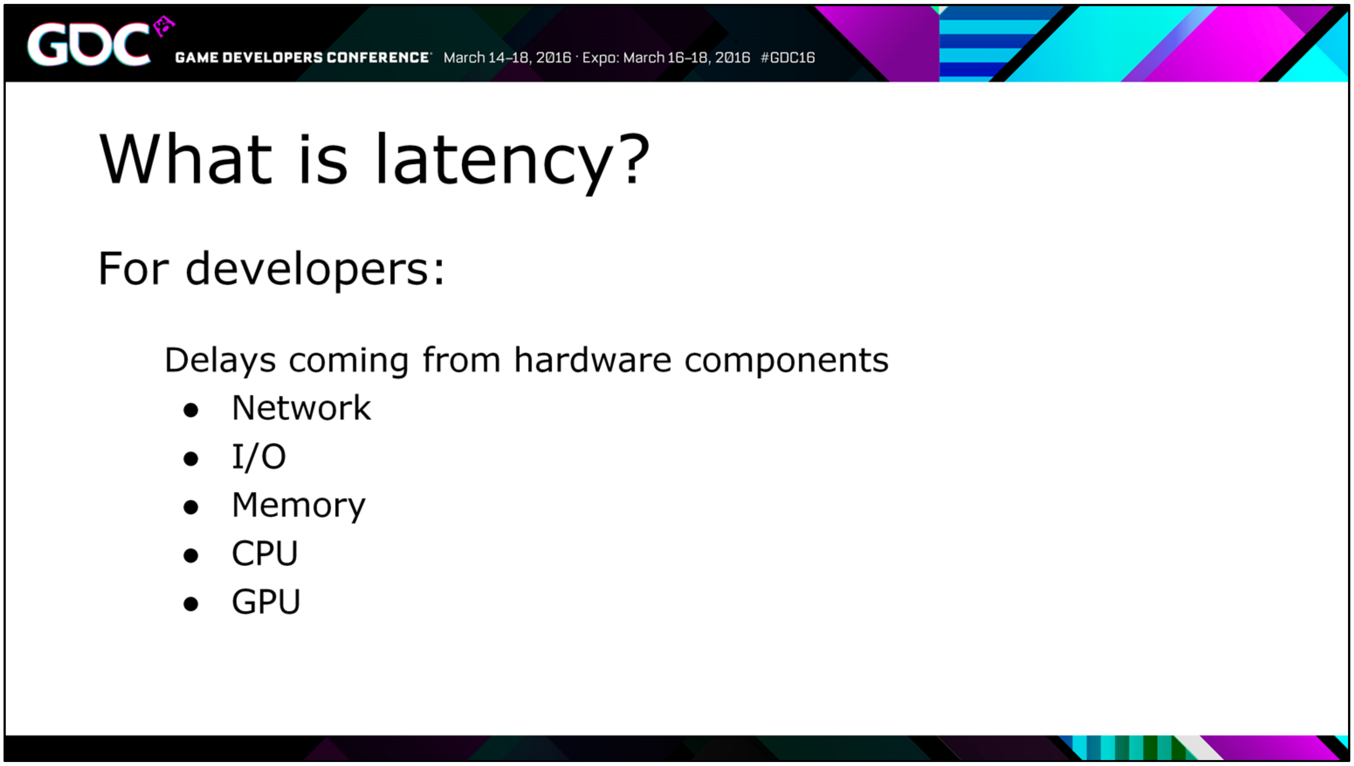
- Pre-alpha build testers, December 2014

When I started working with Treyarch, the initial problem was this.

It was a vague problem. It was just a feeling that something was wrong, something was off.

Was it real or placebo? It was difficult to address it because it was hard to prove.

Treyarch wanted fresh eyes on the problem to get a new perspective.



What is latency?

For developers:

Delays coming from hardware components

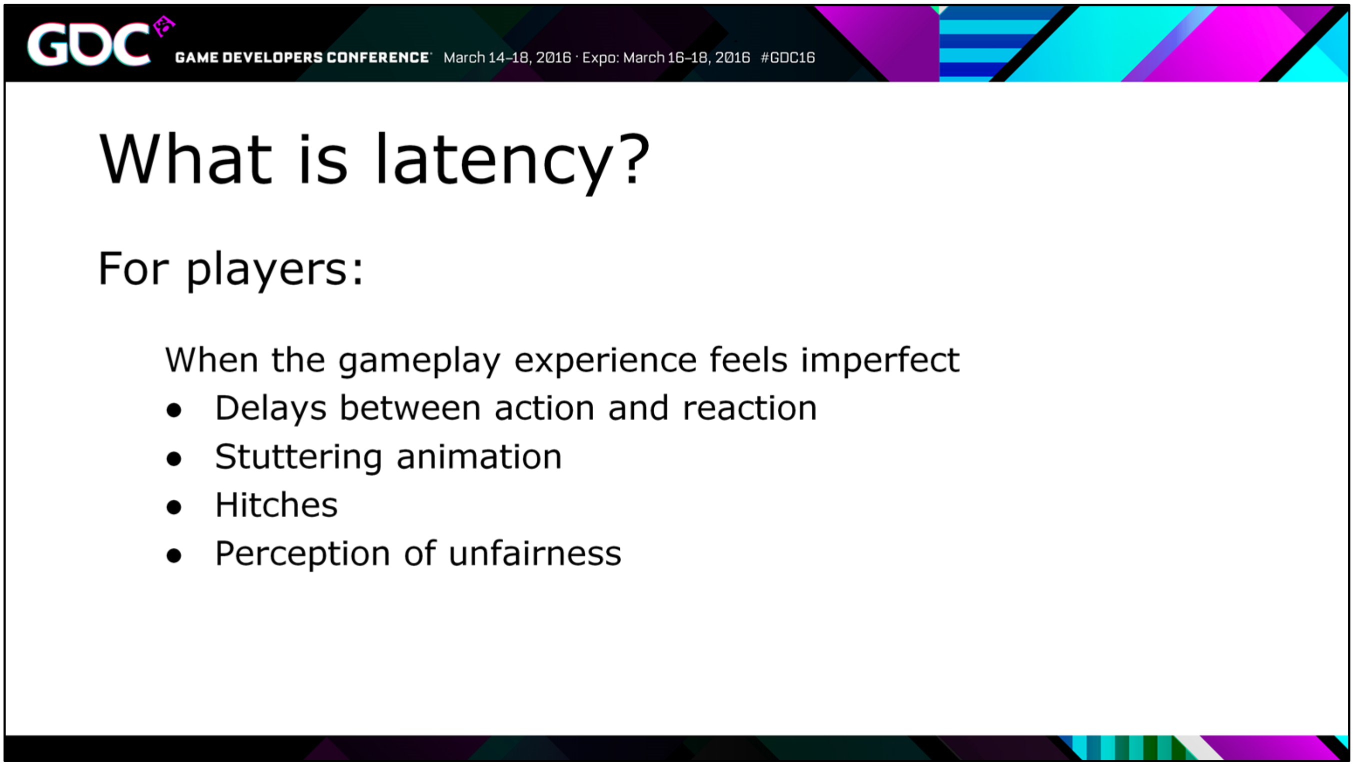
- Network
- I/O
- Memory
- CPU
- GPU

For us it’s a well defined concept.

There’s no magic – Everything we do takes time.

We have good tools to measure each system separately.

We balance it all to create the illusion of immediate interactivity on screen.



What is latency?

For players:

When the gameplay experience feels imperfect

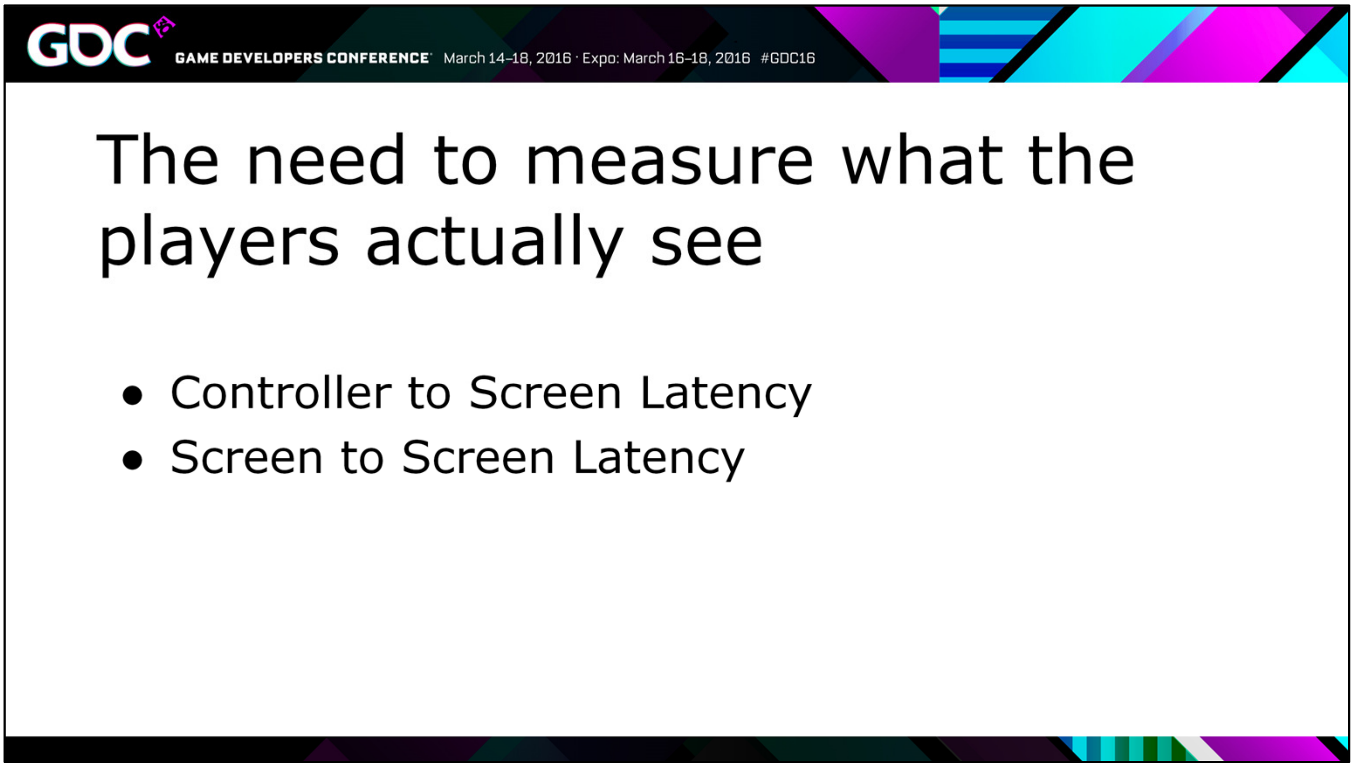
- Delays between action and reaction
- Stuttering animation
- Hitches
- Perception of unfairness

But for players it’s a broader concept.

They call it “lag”. It’s when the gameplay experience is imperfect.

Not just about ping.

Because of this dicrepancy, we can get in a situation where all of our profiling tools are telling us that our systems are okay, but in the playtest the players are complaining about latency.



The need to measure what the players actually see

- Controller to Screen Latency
- Screen to Screen Latency

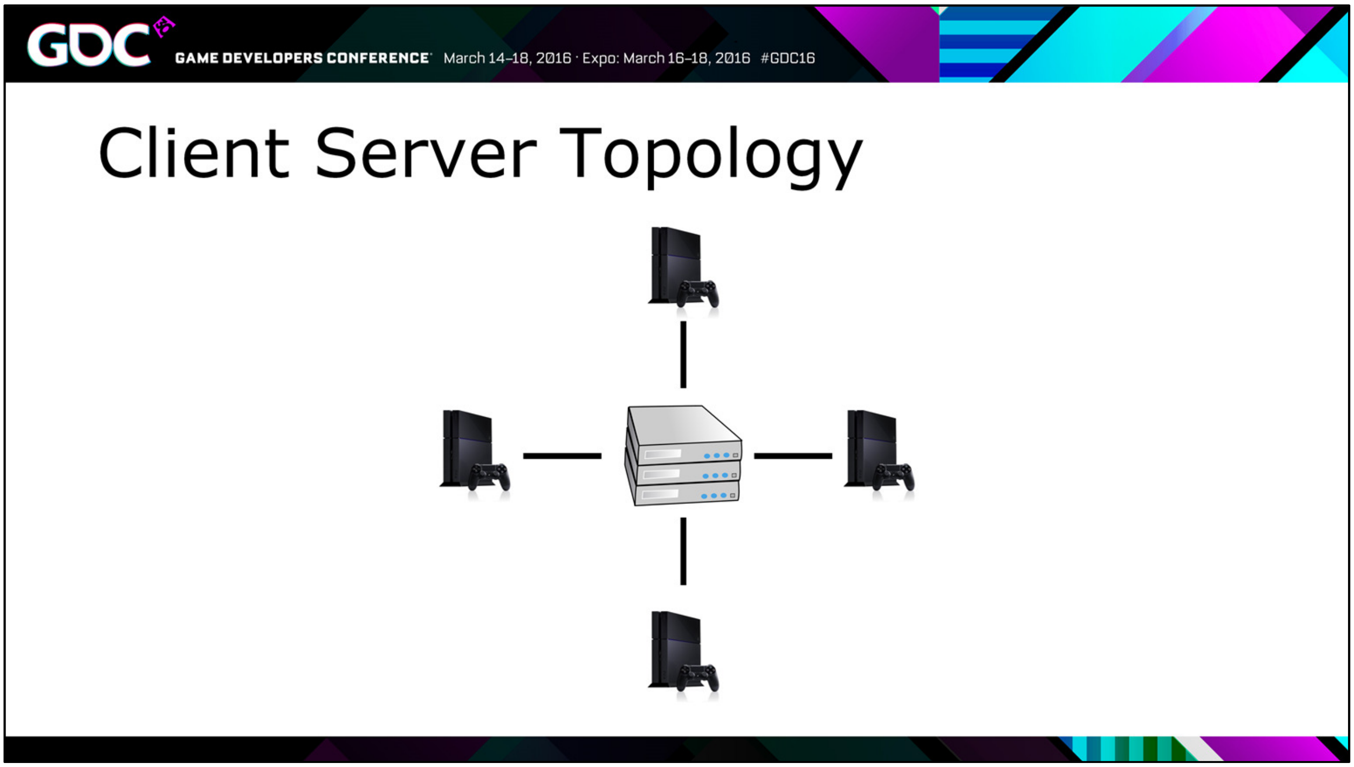
Let’s do a Mythbuster-style investigation!
Mythbuster is a TV show where they often blow stuff up and look at what happenned in slow motion.

Let’s use an high-speed camera to record the screens in multiplayer.

If the players really perceive latency, we should capture it.

In this talk

- Current tech & tools for fighting latency
- Camera setup
- Measuring screen to screen latency
- Measuring controller to screen latency



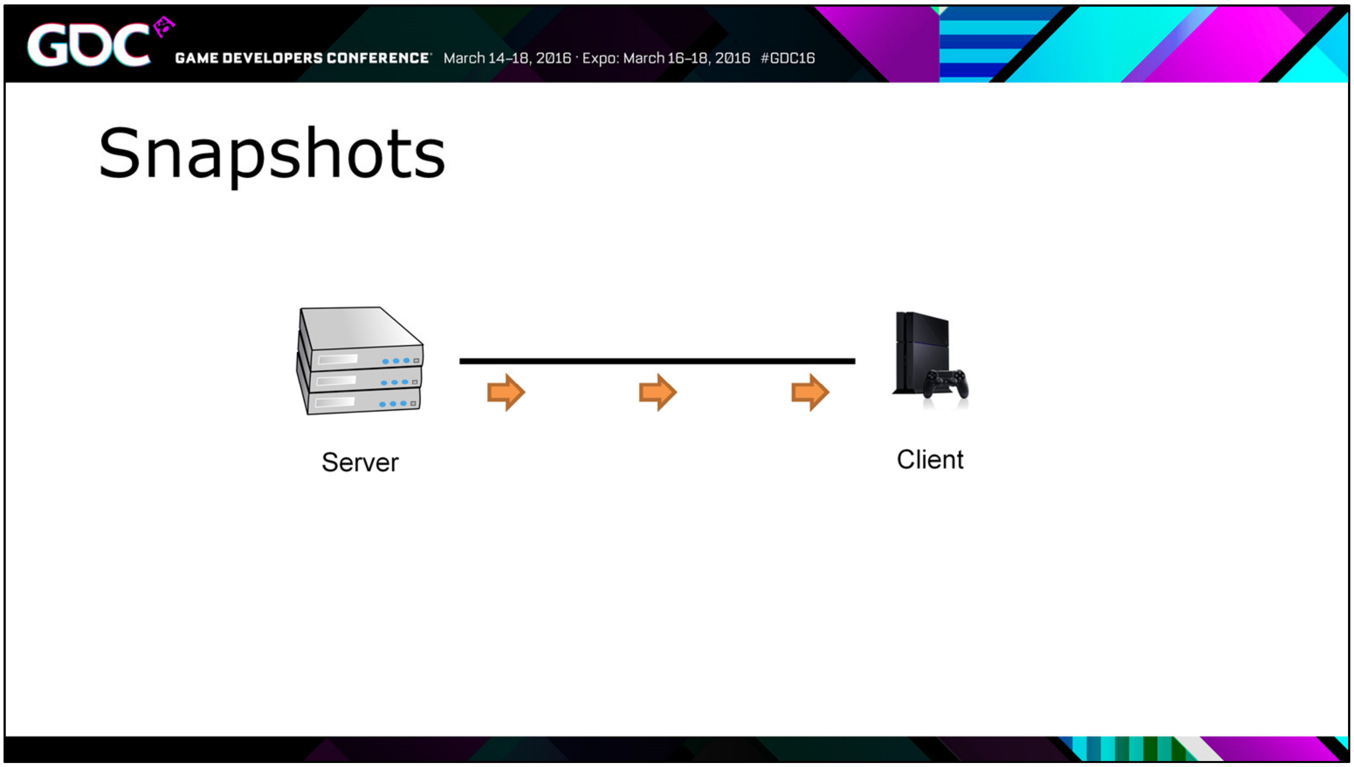
Client Server Topology

Clients are connected to a dedicated server hosted in data center.

The server is 100% authoritative of the gameplay state.

Consistent state for all players (no divergence other than your own character due to client-side prediction).

But not as direct as a peer-to-peer connection.



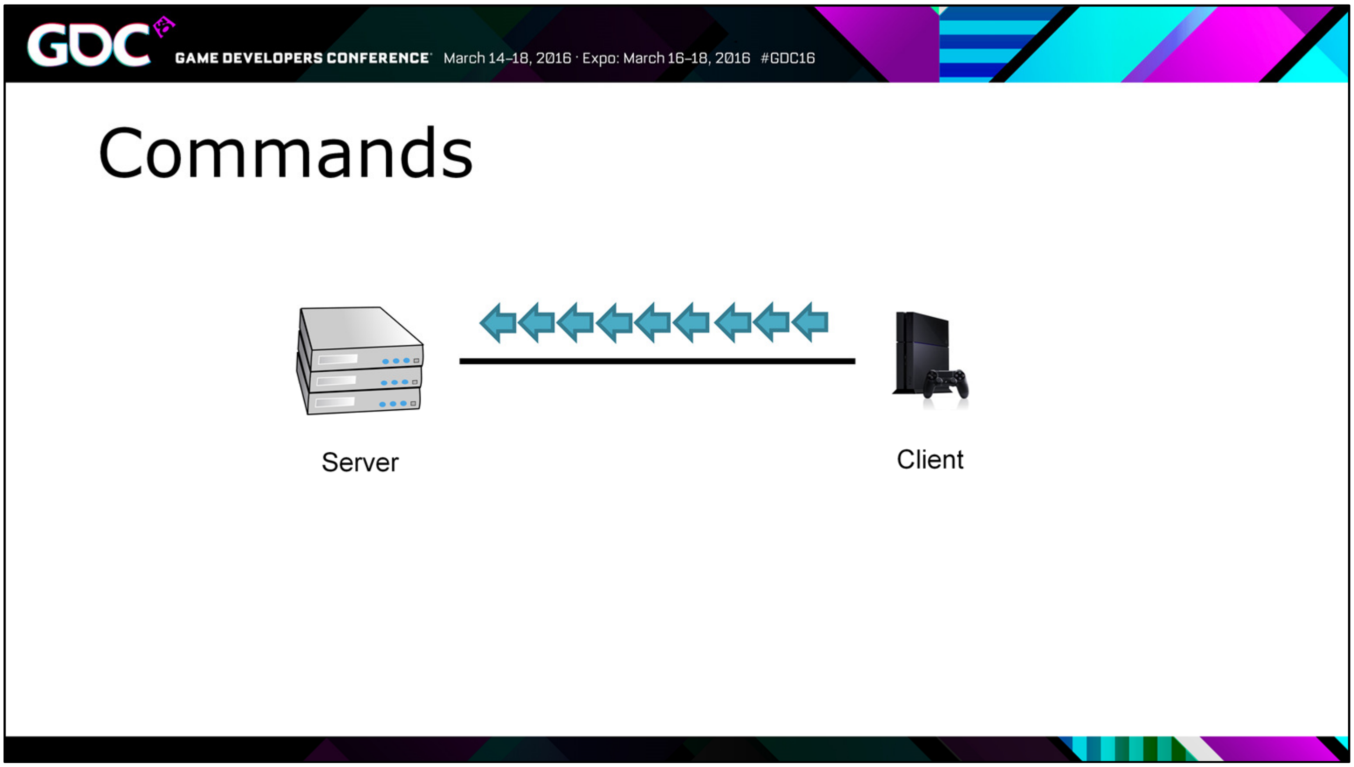
Snapshots

Server sends updates (called snapshots) at a fixed rate.

A snapshot contains everything about the game state.
They are delta compressed against the last acked snapshot to reduce size.
They usually fit within the MTU of a UDP packet.

Using UDP because we don’t need a reliable transmission.
It’s actually better to not resend a drop packet because we already have a fresher one in flight.

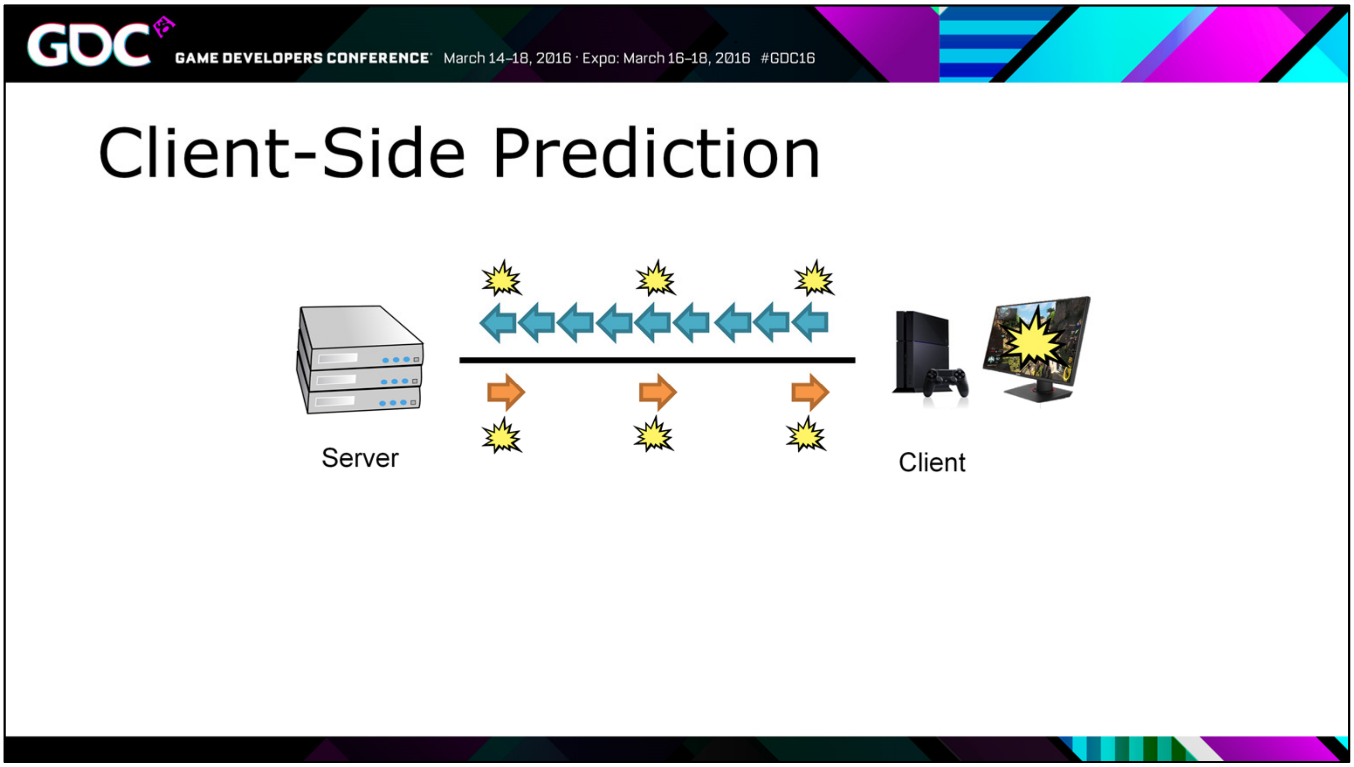
The snapshot rate is slower than the frame rate and requires interpolation to have smooth animation.
Interpolation introduces a little bit of latency, as we wait for a second snapshot to arrive.



Commands

The client intentions are sent to the server in a command packet.
Very small UDP packet sent every frame.

Reliability is important, so all un-acked commands are re-sent at every packet – just in case there’s a dropped packet.
Sacrifices bandwidth for better latency.

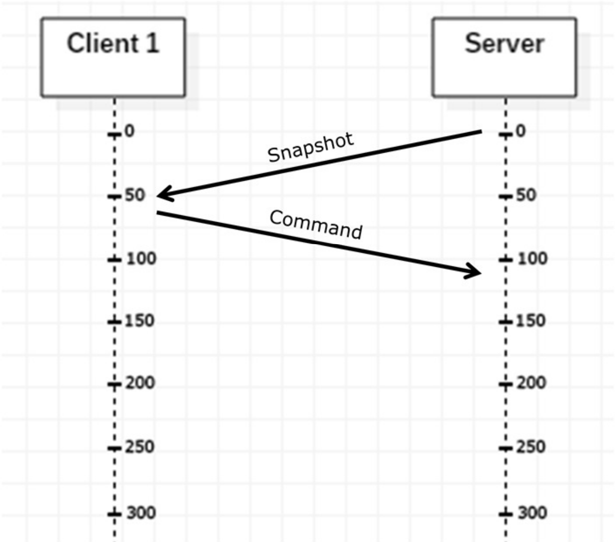


Client-Side Prediction

For input responsiveness, apply the local inputs immediately – Predicting the state the server would send us.

Waiting for the round-trip would add unacceptable latency for a shooter game. Even on LAN.

Every shooter do this.

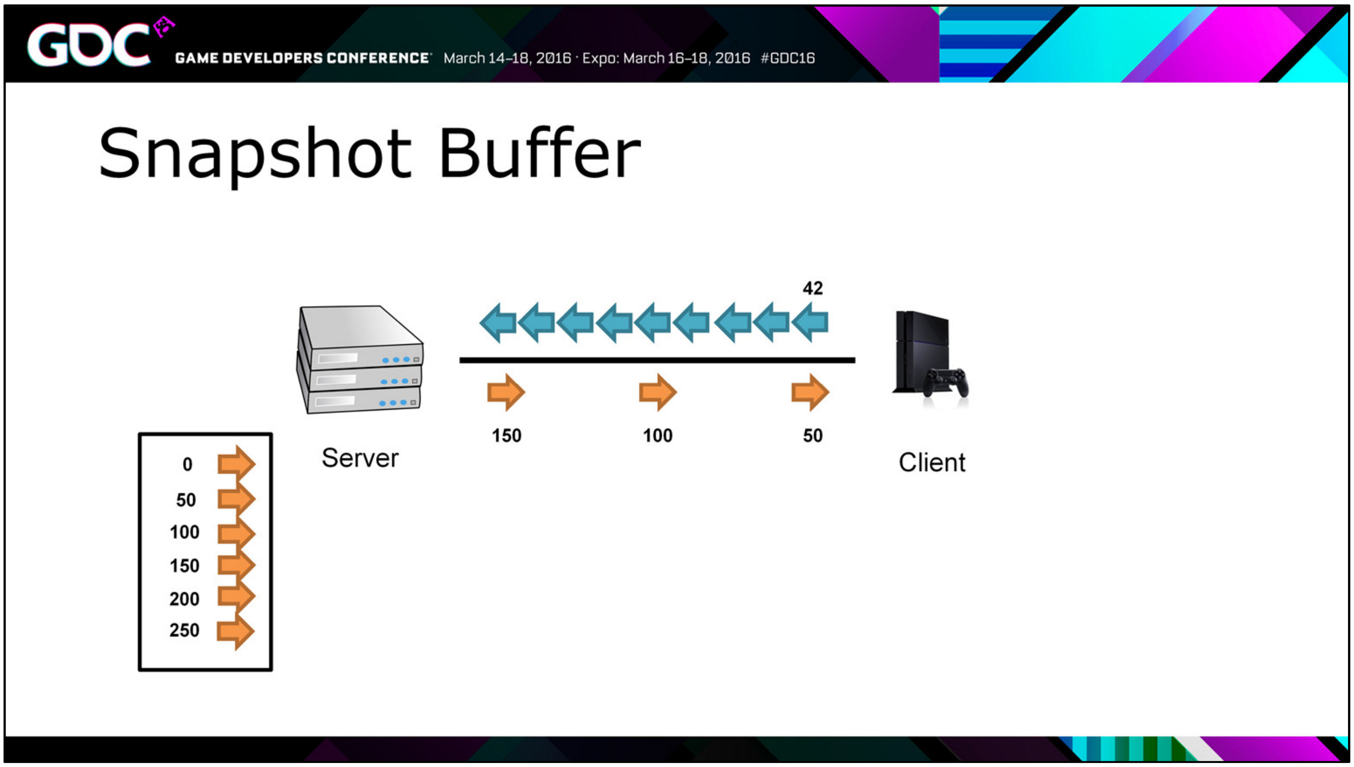


The artifact of client-side prediction.
Because it always take time for packets to travel, what a client is looking at is always the past. But he can move his player locally before everyone else knows about it.

“You are always looking at the past from a viewpoint in the future”.

And everyone is at a different time in the past / future.
This is the kind of absurd time-travel non-sense that network programmers have to deal with.
We balance it all to create the illusion that everyone is playing in the same world at the same time.

UML Sequence Diagram – For showing interactions between 2 systems.
Twist: Using diagonal arrows to represent latency.



Snapshot Buffer

42

150 100 50

Server

Client

0
50
100
150
200
250

The server keeps an history of previous snaphots. To improve hit-detection.
Because if you shoot at a fast moving target at frame 42, by the time your command packet is processed by the server, that target won’t be there anymore. And the hit will miss. Even on LAN conditions.

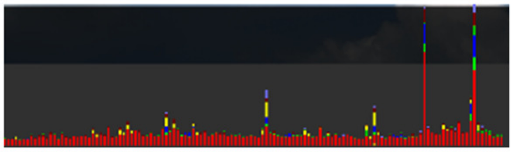
The server is able to re-calculate where players were at frame 42 when processing your shot, because he’s using the same snapshots (0 and 50) and the same interpolation algorithm the client was using.

Players can focus on shooting at what the see on their screen.

Debug Tools

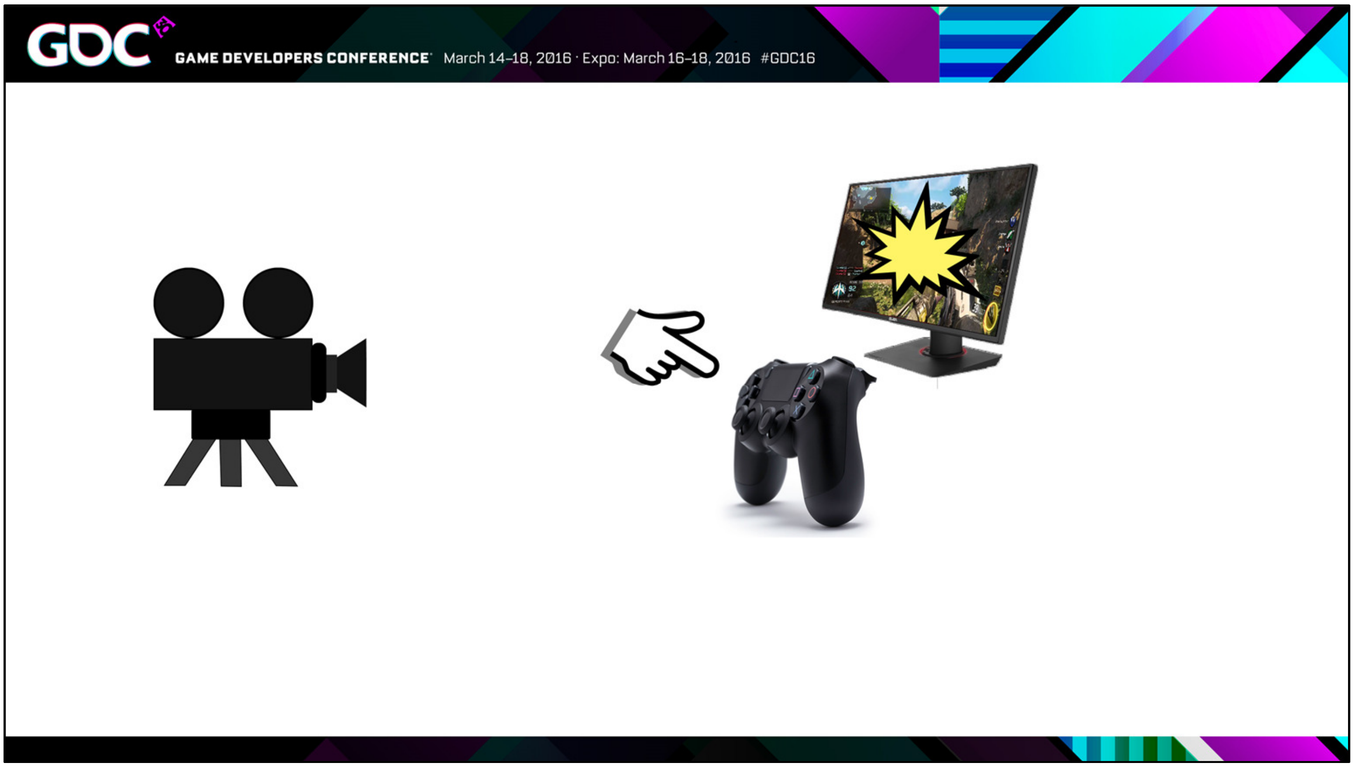
- CPU Profiling
- Network Profiling & Shaping
- Log Traces & Asserts
- Video Recording (single screen)

Ex: Bandwidth Monitor



Classic tools used for debugging latency issues.

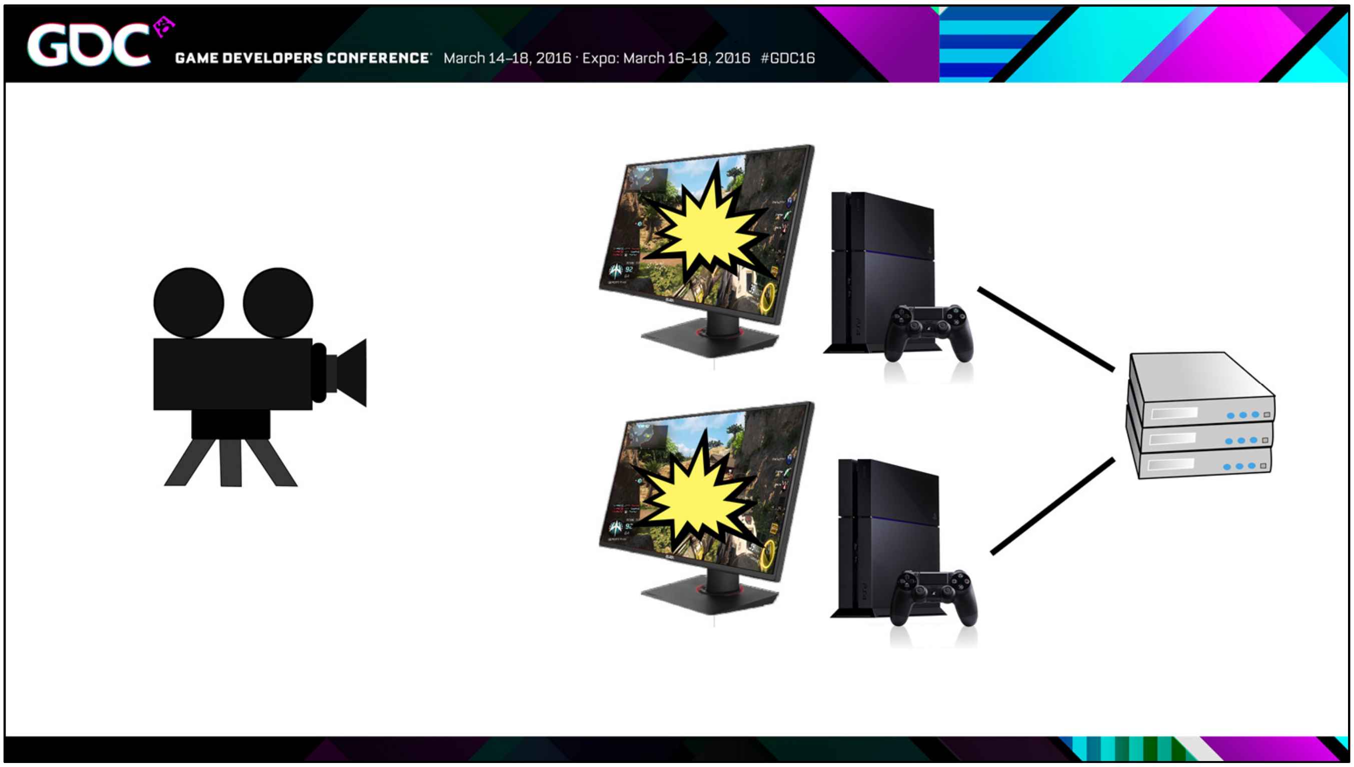
Bandwidth can also introduce latency because a spike in bandwidth will cause buffering and fragmentation which can cause delays.



The concept for measuring Controller to Screen Latency:
Record, with a camera, a button press and the reaction on screen.
Then, looking at the video file, count the frames between the two events.

For example, with a 200fps camera (5ms per frame), 9 frames of video means 45ms latency.

We've been doing this for a long time at Activision for measuring controller responsiveness.



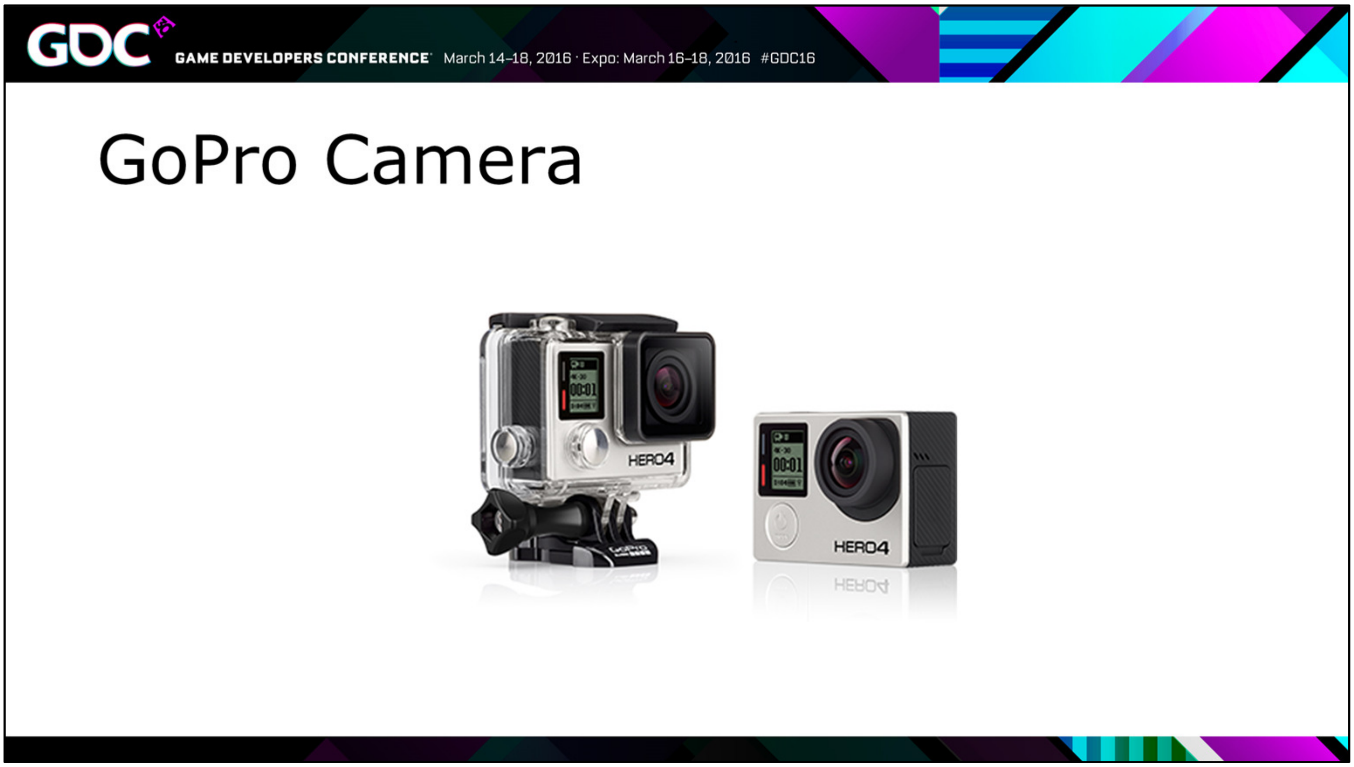
Using the same concept, but for measuring multiplayer latency. Screen to screen latency.

Why not record both video feed and line up the times?
Because syncing the network time is a netcode feature. So we would be measuring the accuracy of the netcode using the netcode.

The camera is a neutral observer. It is capturing in its own time referential two events happening in separate time referentials.

The camera choice is important.
Need a high-frame rate but also a good resolution to be able to see two 1080p screens side-by-side.

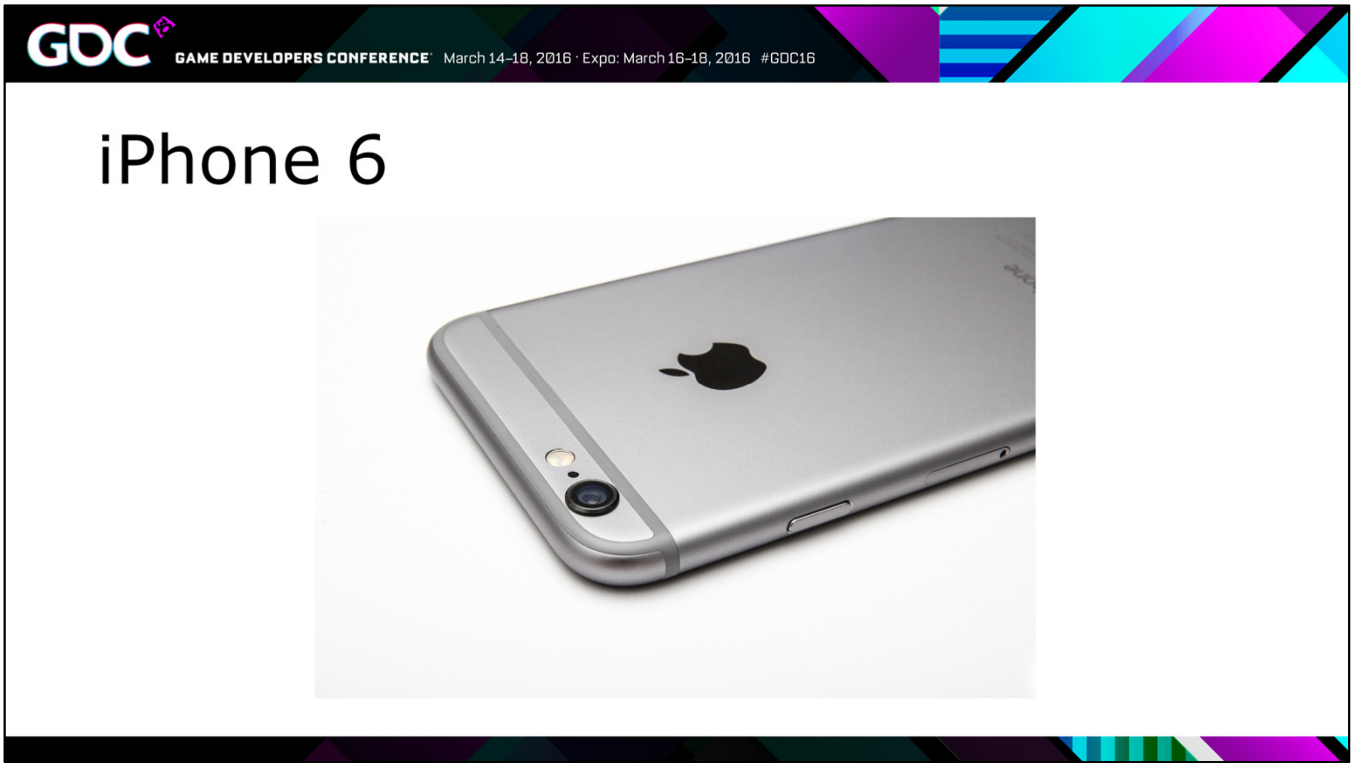
There were not many consumer-grade camera on the market at that time (2014) that could do this.



GoPro Camera

I first tried with my own GoPro Hero4 Black.
Settings used: 1080p, 120fps, Protune ON, Low Light OFF, Spot Meter ON, ISO 1600, EV COMP +1.0
The original firmware back then did not allow to reach 240fps.

It worked. But the resolution was lacking. Also lens distortion and image noise in low light were issues.



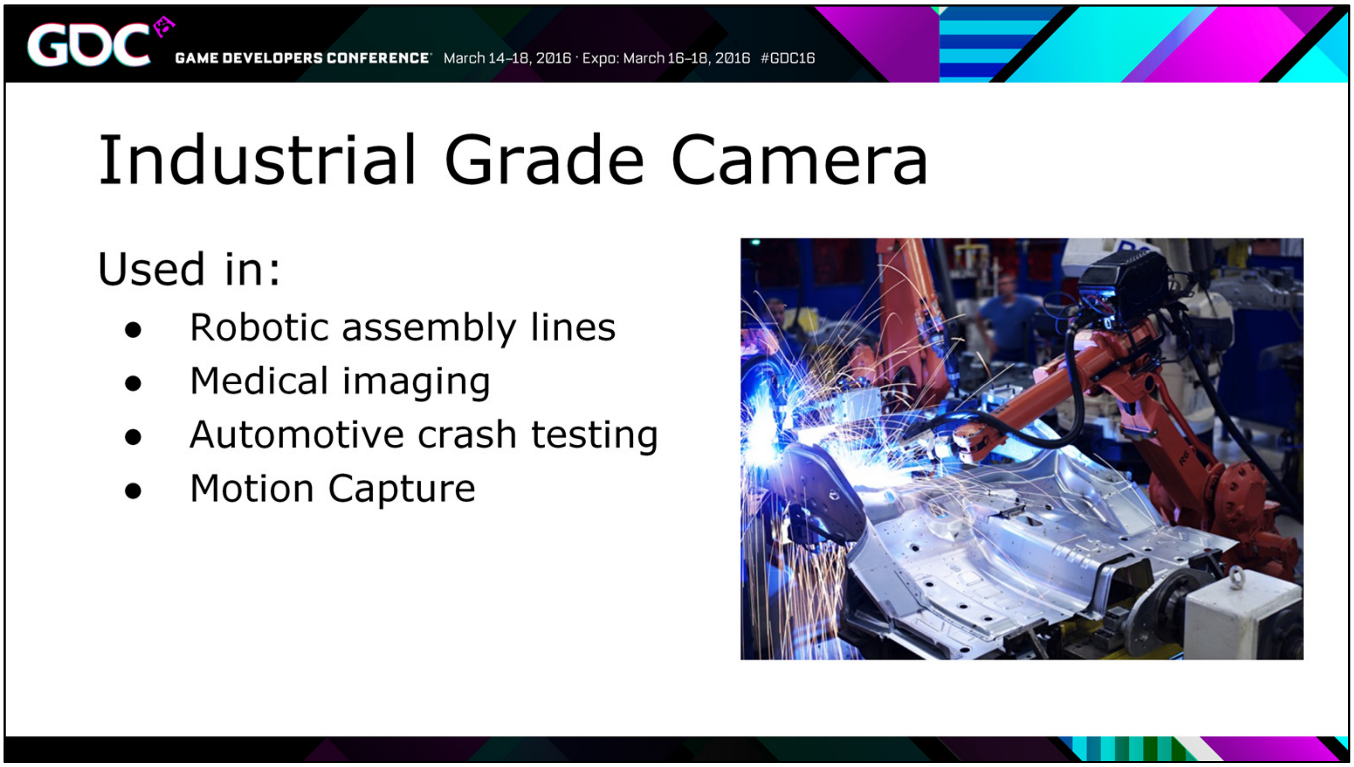
iPhone 6

iPhone 6: 240 fps at 1080p
Works better than the GoPro for this kind of test.

Ximea xiQ

Borrowing a camera from our Mocap studio (thanks Javier!!)

The camera is a Ximea xiQ
1“ sensor 4.2 MP
2048 × 2048 pixel
Greyscale



Industrial Grade Camera

Used in:

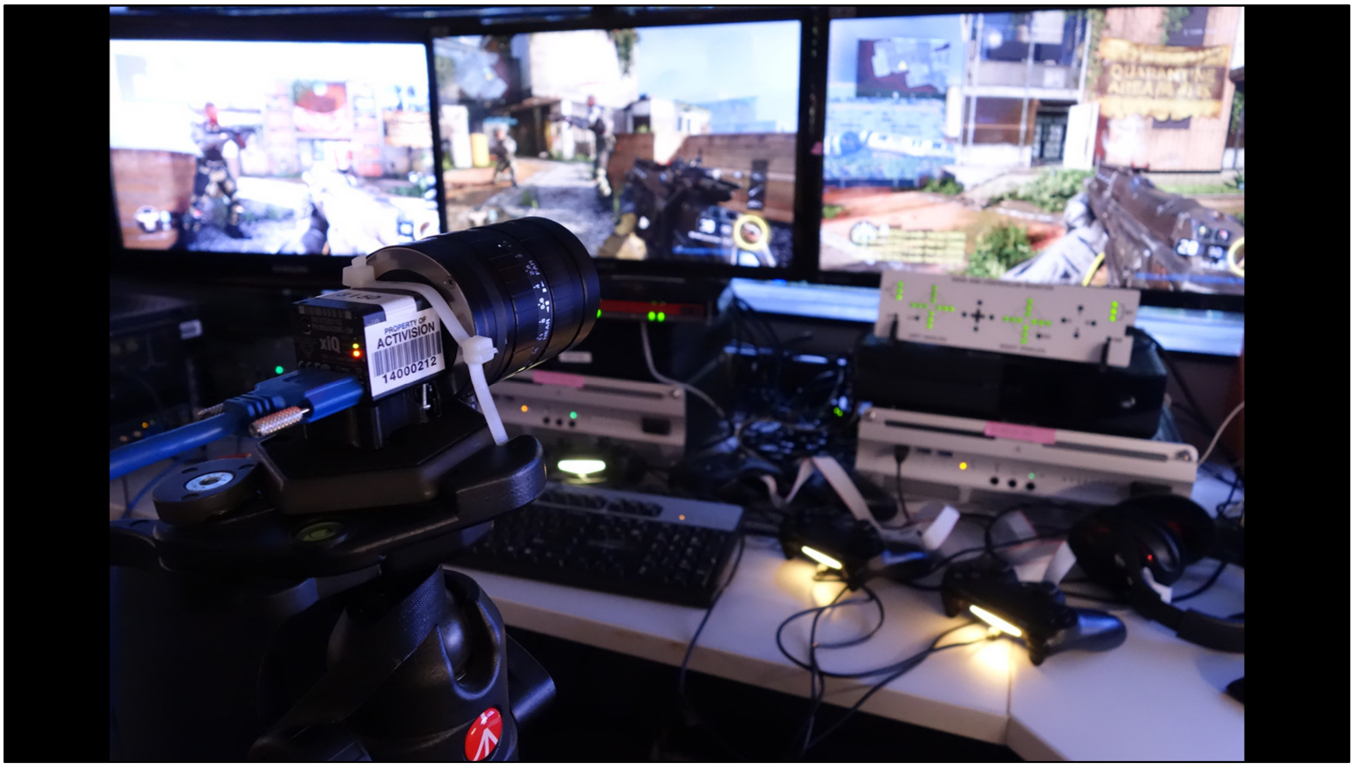
- Robotic assembly lines
- Medical imaging
- Automotive crash testing
- Motion Capture

“Legit” camera. Good for capturing motion with a high-fidelity

We use it in our performance capture rig.

The camera is basically just a sensor and an API. No camera body.

Needs a lens and a recording PC connected via USB3.



5Gbps data rate, but in practice I was getting about 1Gbps. 85X the rate of the GoPro Hero4 Black.

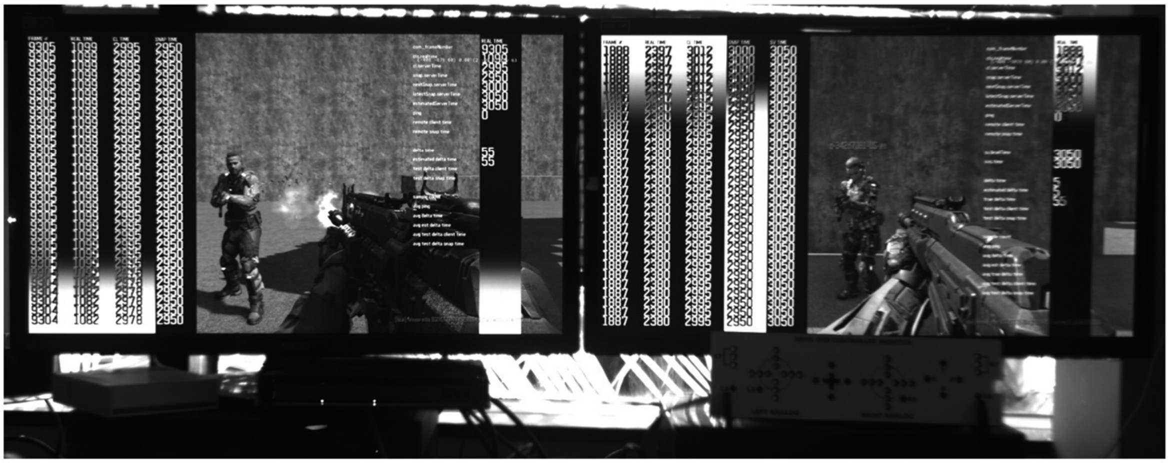
Limited by the speed of the recording PC:
USB3 PCI-E card and dual SSD drives.
200fps at 2048x800. Which is a good aspect ratio for two 1080p screens side-by-side.

Using an in-house recording software from the mocap studio.
Recording in a loss-less greyscale CODEC.
Huge amount of data for a few seconds of recording.

Can adjust all the manual settings, such as exposure time.
Outputs an XML file with the timing of each frame. Which is monumental because the foundation of this technique is to know how long a video frame lasts. The consumer-grade camera can sometime capture more light (longer exposition) to compensate for low-light situations. Even if the resulting file is at a certain fps, we’re not sure that’s true.

Important to have two identical monitors – Otherwise you measure the difference between them.

Counting Frames

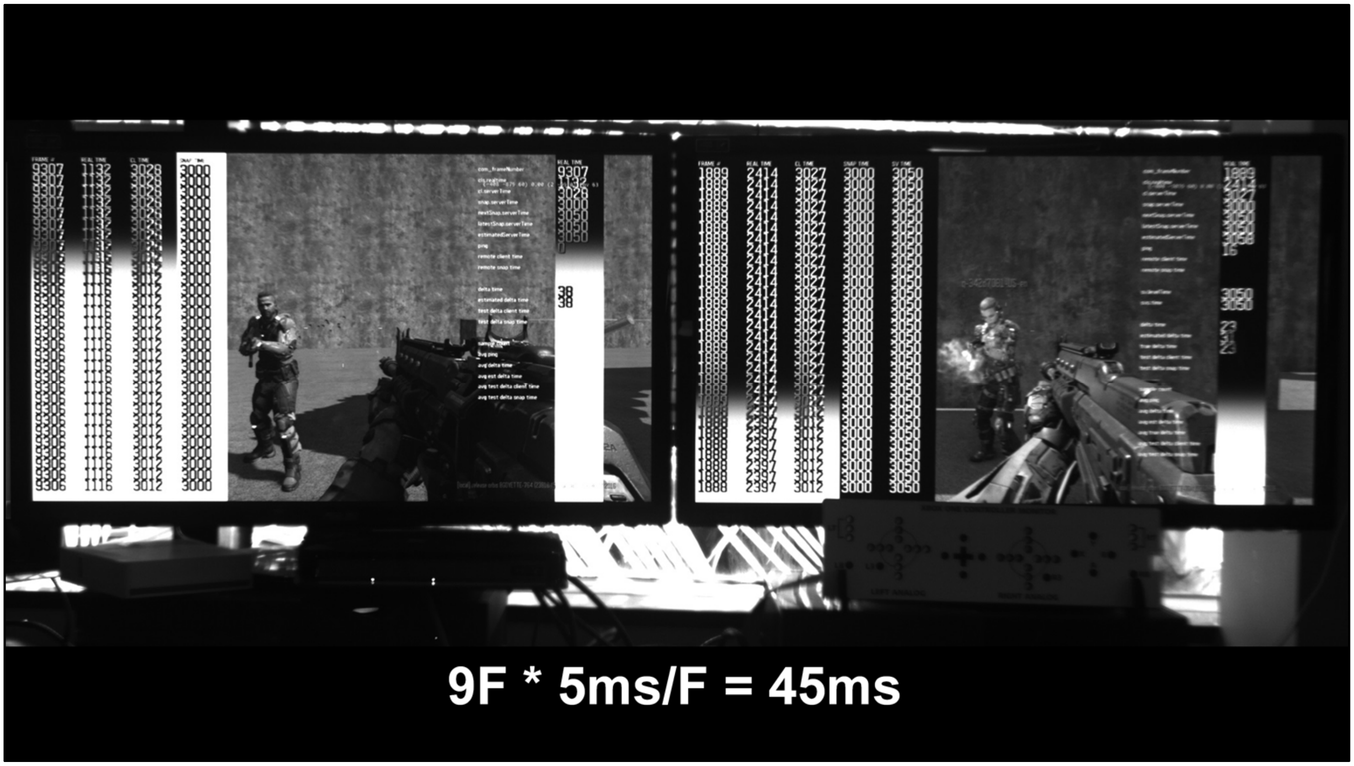


The tedious part of the work.

Drawing High Contrast Timing Bars for readability.
Displaying relevant info such as Frame Number, Client Time, Snapshot Time, Server Time, etc.
Change the background color black/white each time the number changes so that we know what we are looking at.

We see the monitor scanning the screen. Getting 3 frames of video per game frame.

Video – Shooting latency, full speed



9F * 5ms/F = 45ms

Animation – Stepping frame by frame

Find the first frame where the player starts shooting on the left (1st person)
Count the frames until he starts shooting on the right (3rd person)

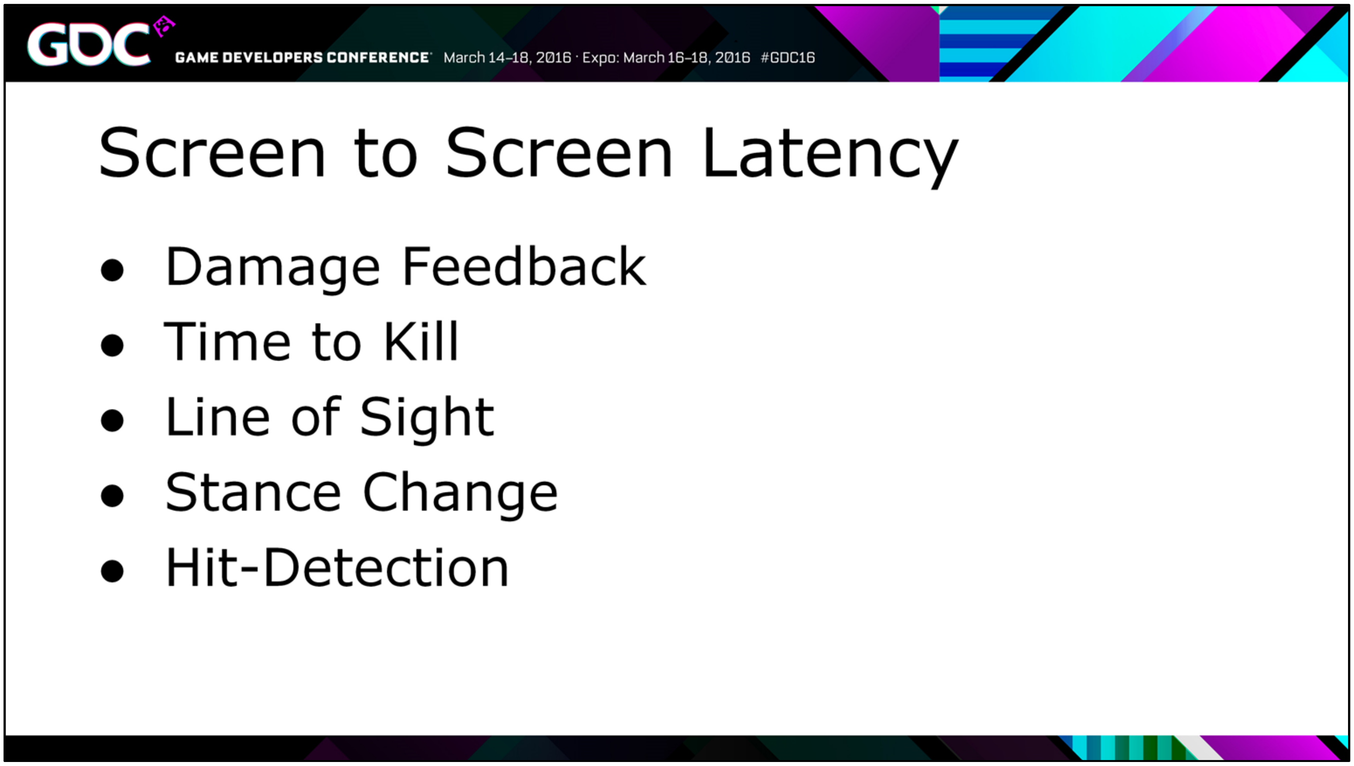
Always take the measurement at the same location on the screen (middle). Even if you know the frame being drawn will be the one, keep counting until the middle of the screen is refreshed.

In this case we measured 45ms base latency (or 2.7 game frames)

Repeat about 50 times. Calculate the Average, Min, Max, Variance. Test various in-game actions to see if one is off: Jumping, Firing, Standing to Moving, Etc.

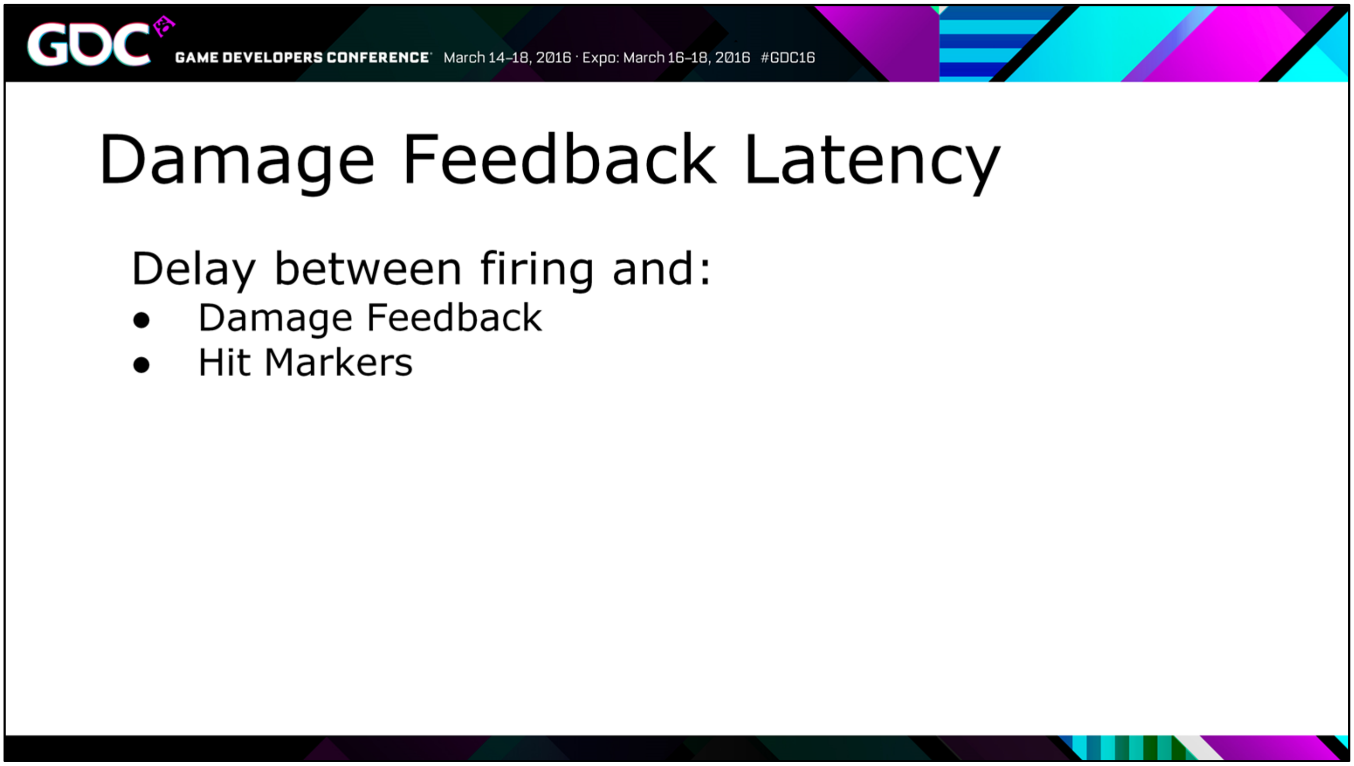
Always keep detailed information about the environment of your test. You never know when you'll need look at past results to verify a new hypothesis.

- Test date, changelist, build configuration
- Platform, Hardware revision, SDK version
- Map, gamemode, loadout
- Network LAN/Online, emulation settings
- Controller, wired/wireless, USB Mode
- Monitor model, resolution, refresh rate
- Camera model, frame rate, exposure time
- Special notes, console variables changed, etc.



Screen to Screen Latency

- Damage Feedback
- Time to Kill
- Line of Sight
- Stance Change
- Hit-Detection



GAME DEVELOPERS CONFERENCE March 14–18, 2016 · Expo: March 16–18, 2016 #GDC16

Damage Feedback Latency

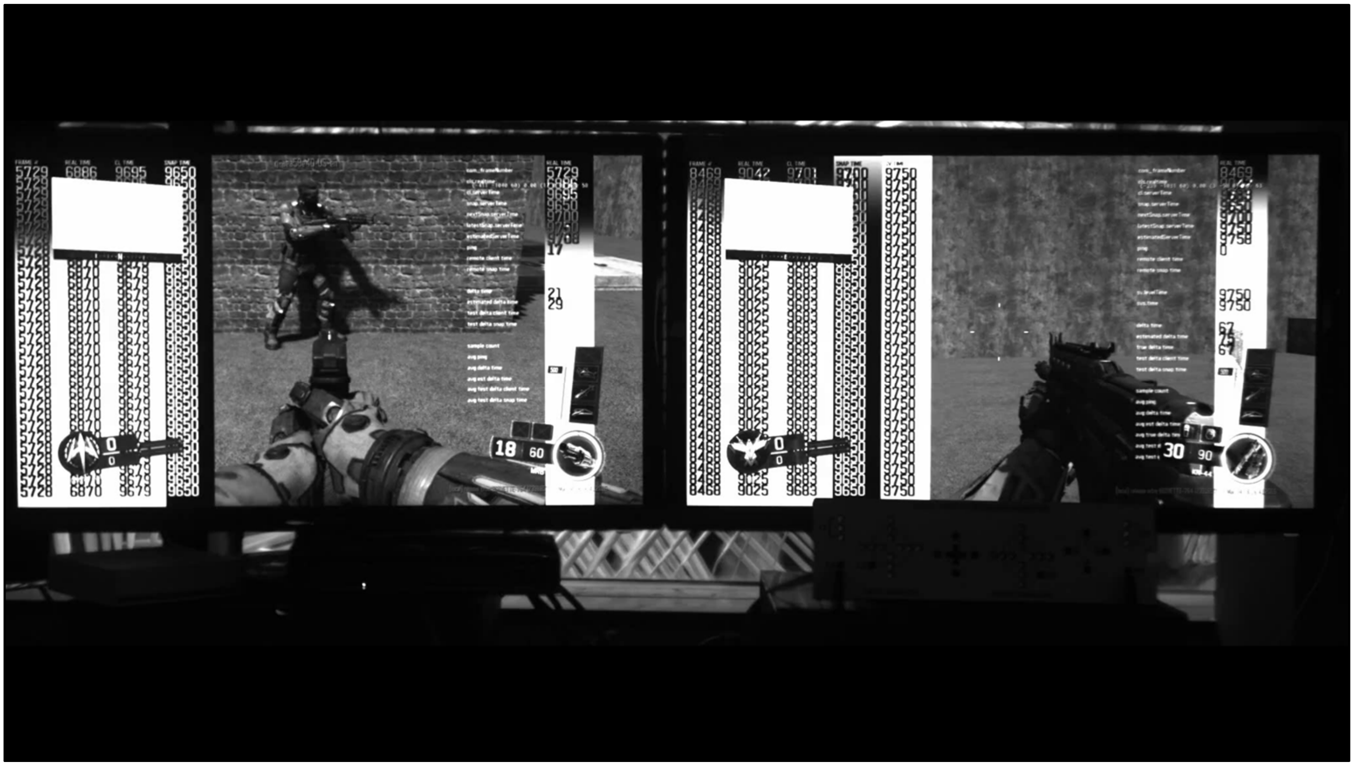
Delay between firing and:

- Damage Feedback
- Hit Markers

Why is it important?

Having latency in the Damage Feedback can be a competitive advantage for the attacking player. The target will have less time to react.

Having latency in the Hit Markers makes it feel like there’s bad hit detection.

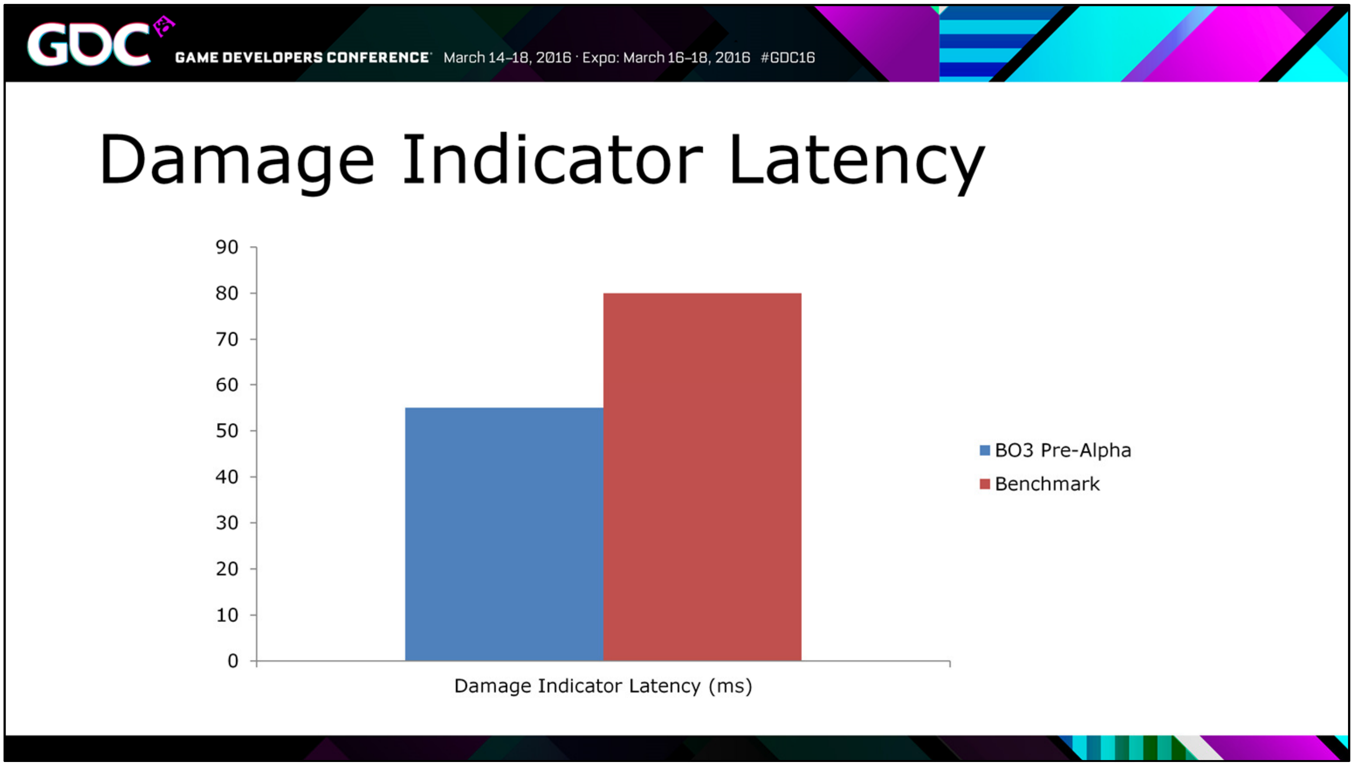


Video – Damage Feedback Latency Test, full speed

Video – Damage Feedback Latency Test, slow motion

Count the frame until the middle of the screen is refreshed.

We were expecting to be faster because of optimizations Treyarch had done.



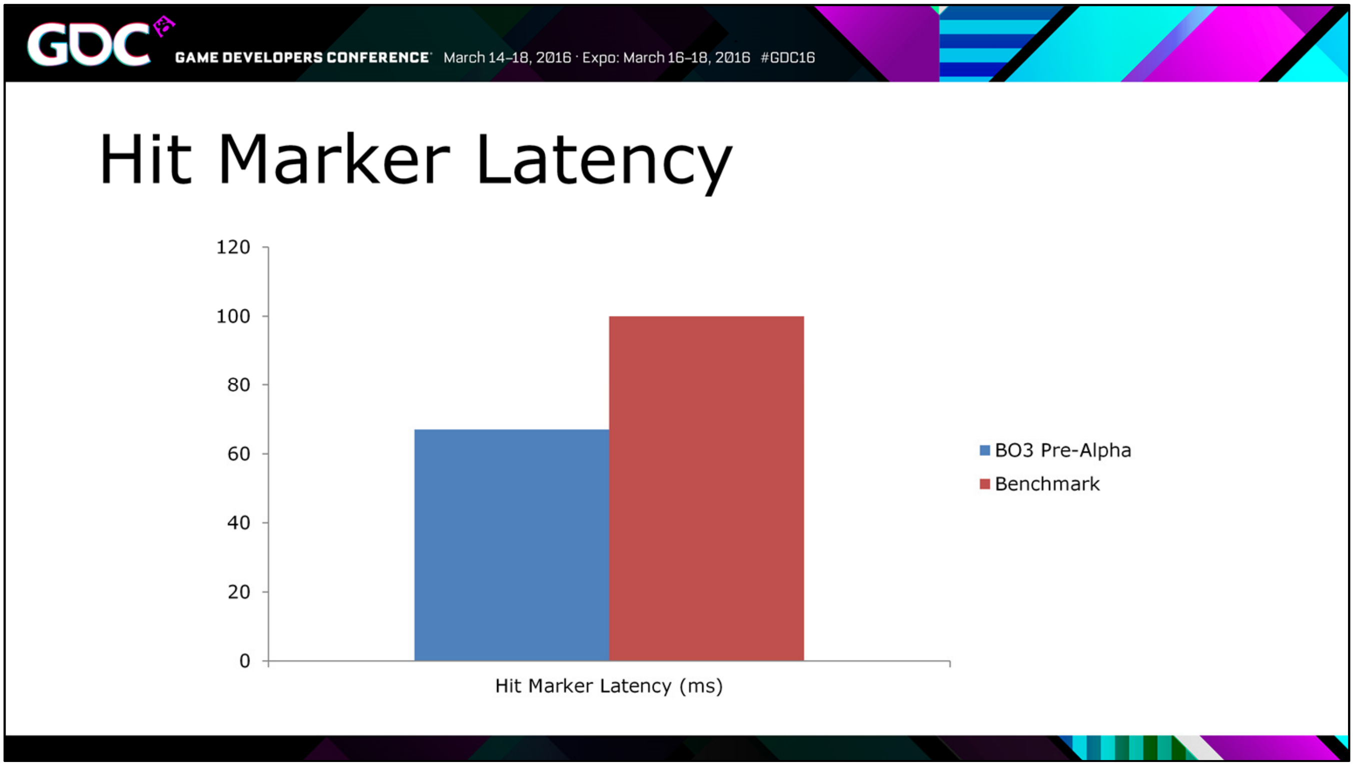
GDC GAME DEVELOPERS CONFERENCE March 14–18, 2016 · Expo: March 16–18, 2016 #GDC16

Damage Indicator Latency

25ms faster

Benchmark: Best measurement from previous Call of Duty games.

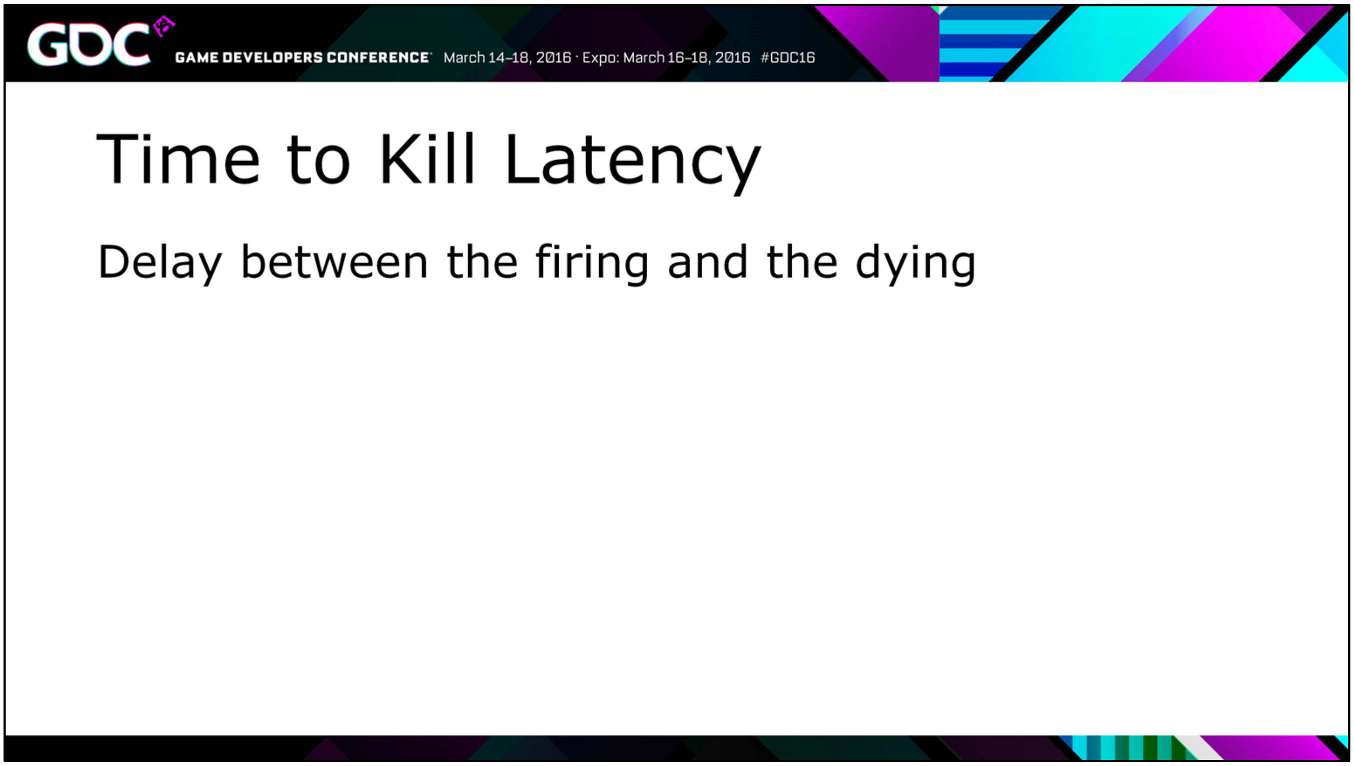
Exactitude VS Precision
It’s important to be able to reproduce the same result when re-measuring with the same equipment. Measuring with different equipment could give a different measurement. The gaps are what is important. The actual latency numbers can be taken with a grain of salt.



Hit Marker Latency

33ms faster

That was good, it was confirming the optimizations the team had done to the engine. But it wasn't explaining the problem we were getting.

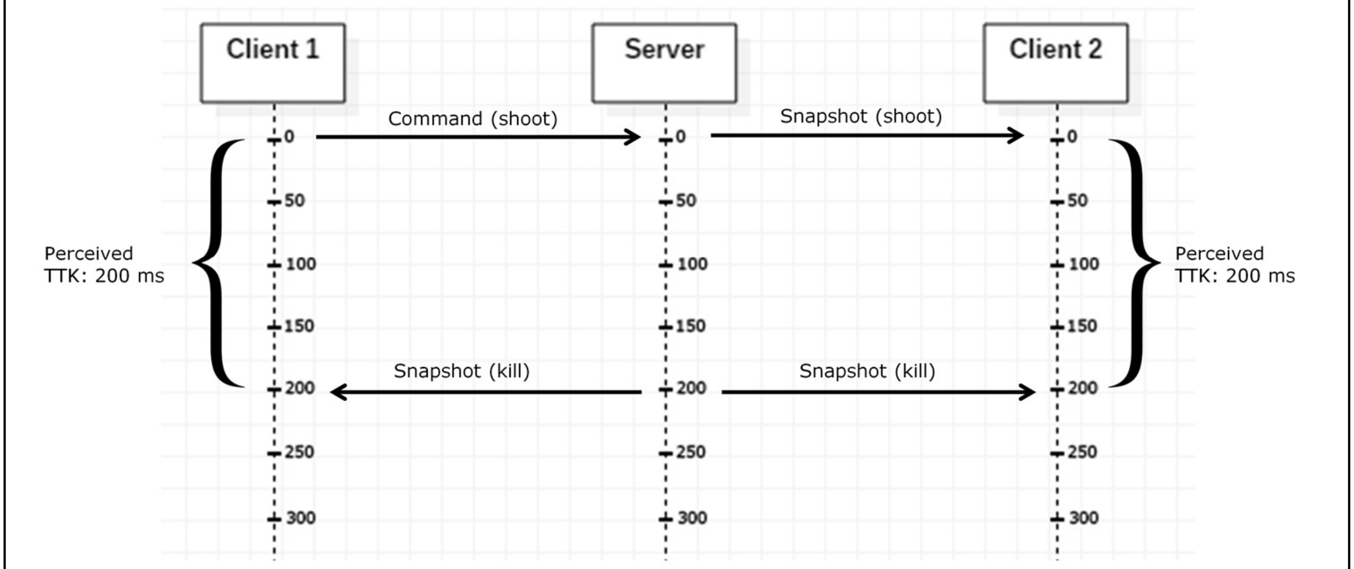


Time to Kill Latency

Delay between the firing and the dying

Why is it important?

For the attacker, a longer TTK feels like the enemies are invincible.
For the victim, a shorter TTK feels like there’s no time to react.



Client 1

Server

Client 2

Command (shoot)

Snapshot (shoot)

Snapshot (kill)

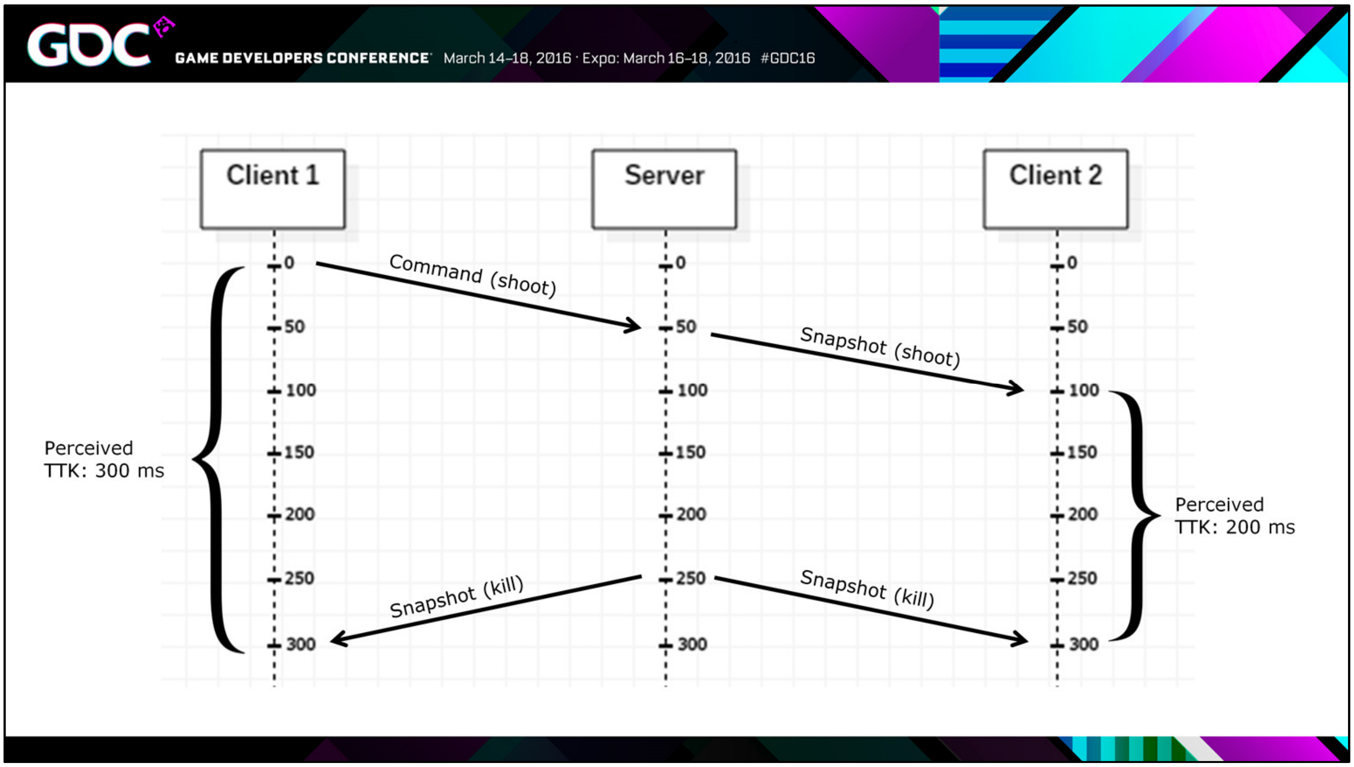
Snapshot (kill)

Perceived TTK: 200 ms

Perceived TTK: 200 ms

In a perfect world without latency, messages would travel instantly.

Both clients, attacker and victim, would perceive the same TTK.



GDC GAME DEVELOPERS CONFERENCE March 14–18, 2016 · Expo: March 16–18, 2016 #GDC16

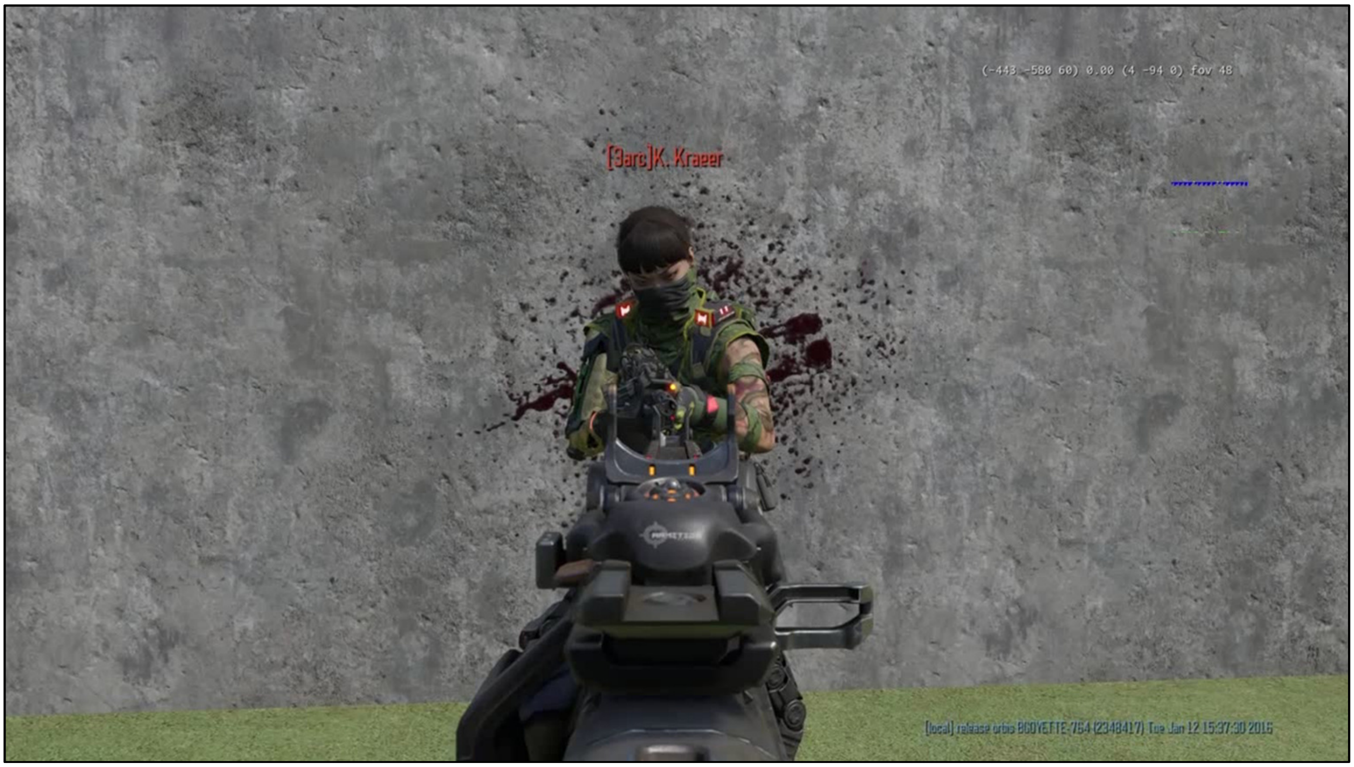
Perceived TTK: 300 ms

Perceived TTK: 200 ms

With latency though, the attacker will feel a longer TTK.
Adding his ping on top of the real value.

This is because we don’t do client-side prediction for the death of players.
Because if we blow up the player and then realize it was a mis-prediction, we would have to bring it back together.
This would look kinda weird!

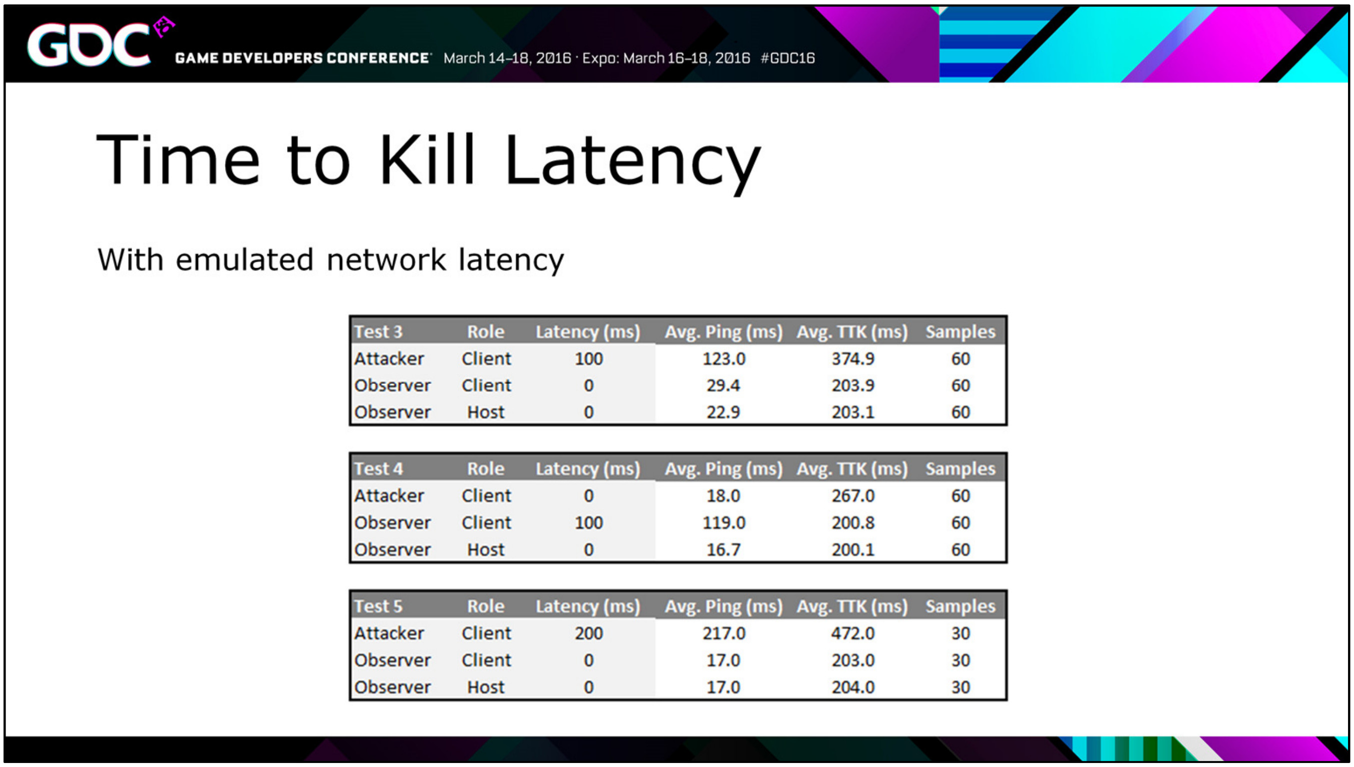
But this was the theory. I wanted to validate it in practice with various network emulation settings. To see when I could break it.


(-443 -580 60) 0.00 (4 -94 0) fov 48
[3arc]K. Kraeer
[local] release orbis BGOYETTE-764 (2348417) Tue Jan 12 15:37:30 2016

Video – TTK procedural tool

Because I would have 1000s of deaths to record, it wasn't going to be practical to count frames.
I did a procedural tool measuring the perceived TTK in-engine. And validated it with the camera. In this case, the test could have been validated with just a single screen video recording, since both events were happening in the same time referential.

Once the tool was vetted, I could iterate fast with various emulated network conditions.



Time to Kill Latency

With emulated network latency

Test 3	Role	Latency (ms)	Avg. Ping (ms)	Avg. TTK (ms)	Samples
Attacker	Client	100	123.0	374.9	60
Observer	Client	0	29.4	203.9	60
Observer	Host	0	22.9	203.1	60

Test 4	Role	Latency (ms)	Avg. Ping (ms)	Avg. TTK (ms)	Samples
Attacker	Client	0	18.0	267.0	60
Observer	Client	100	119.0	200.8	60
Observer	Host	0	16.7	200.1	60

Test 5	Role	Latency (ms)	Avg. Ping (ms)	Avg. TTK (ms)	Samples
Attacker	Client	200	217.0	472.0	30
Observer	Client	0	17.0	203.0	30
Observer	Host	0	17.0	204.0	30

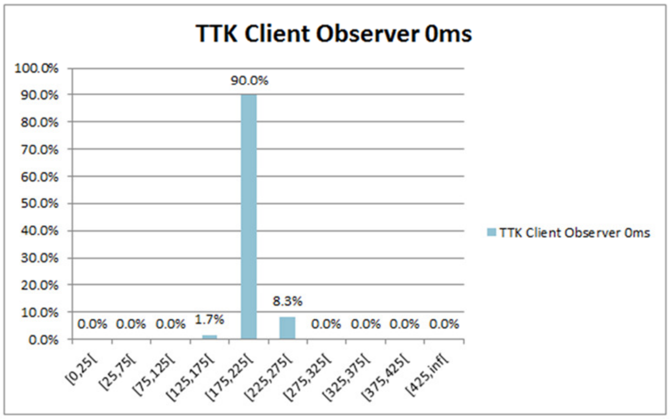
No surprises from the results.

The TTK of the attacker is always longer by 1 server frame + 1 ping.
There’s no difference in TTK whether you’re the victim or observing a kill.

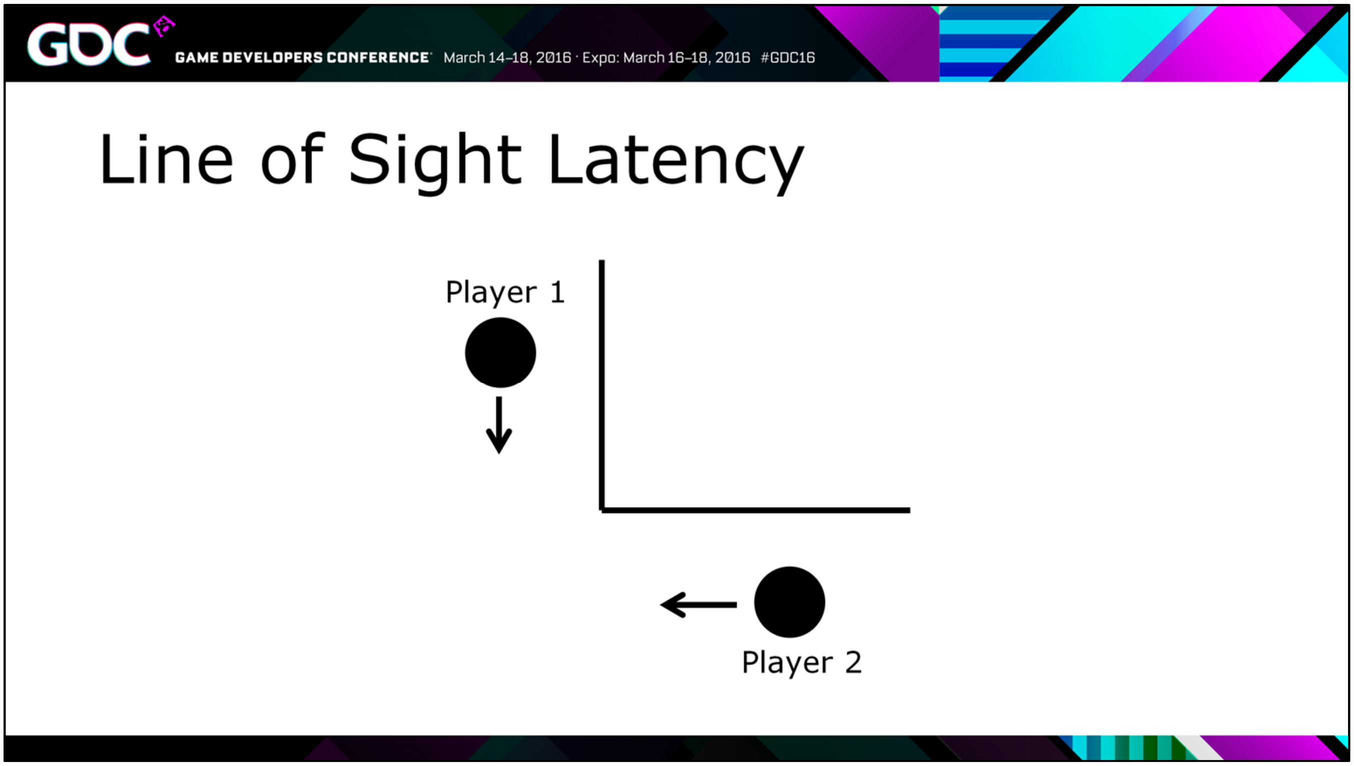
The network conditions start to have a significant impact well beyond the TCR limits.

Time to Kill Latency

With emulated network latency (attacker at 100 ms)



Important not to look just at averages for this test.
Group results into buckets and see how often there are irregular TTK.

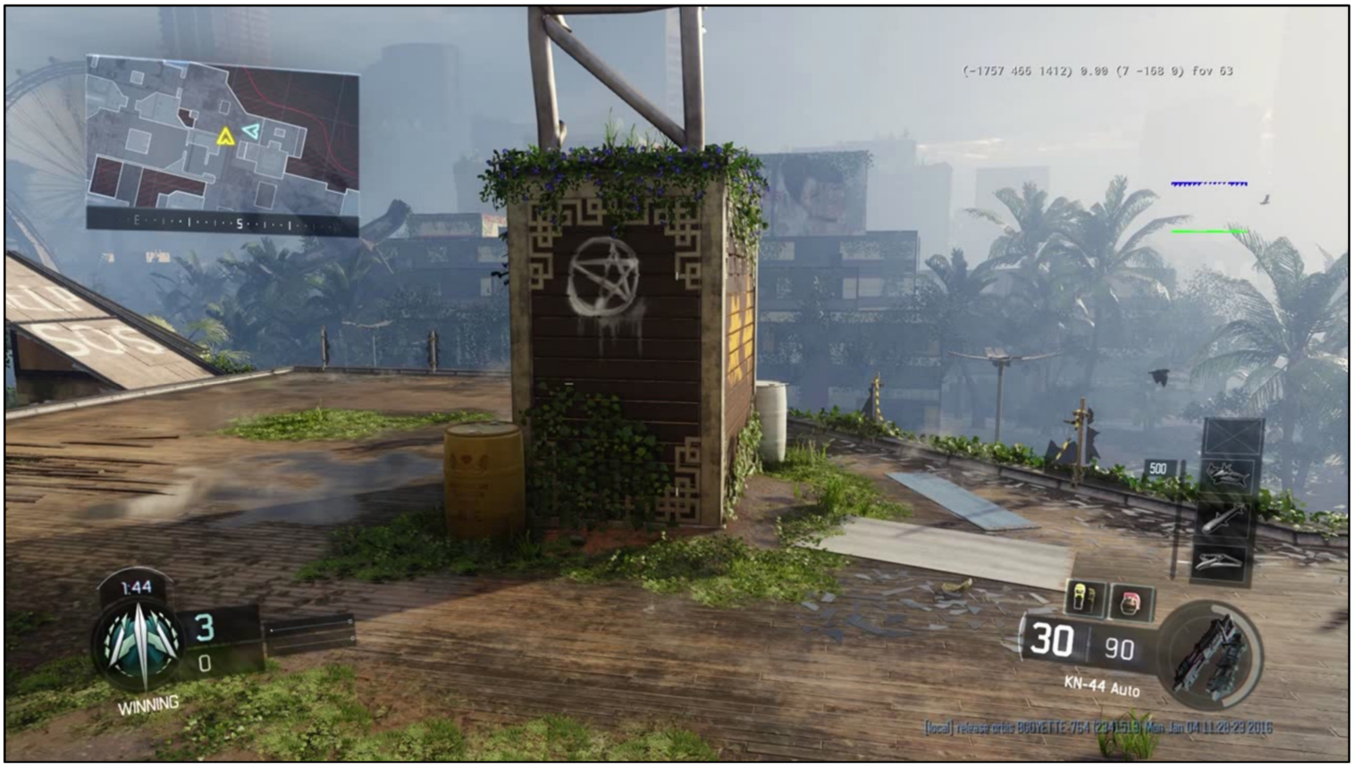


Line of Sight Latency

Delay between “I see you – you see me”

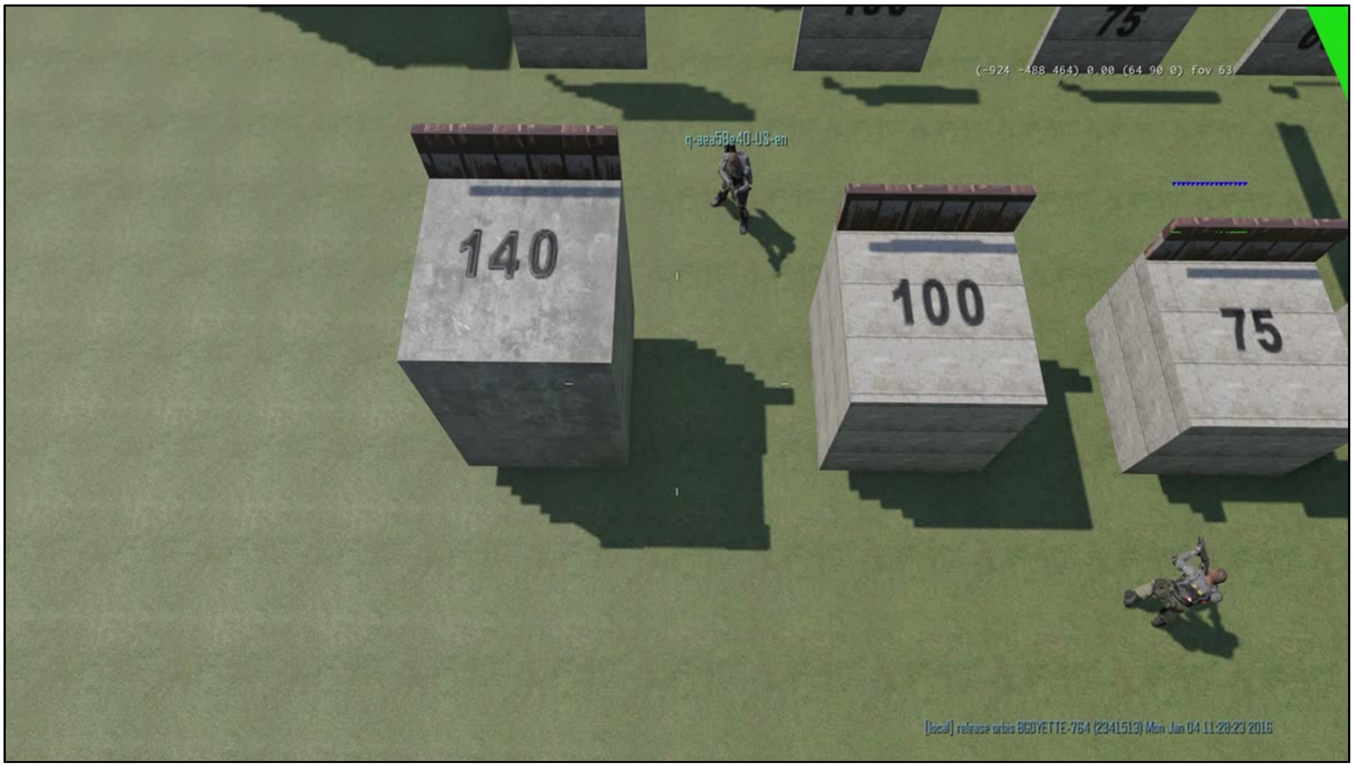
Why is it important?

It’s a competitive advantage to be able to see your opponent earlier.


(-1757 466 1412) 0.00 (7 -168 0) fov 63
E
S
1:44
3
0
WINNING
500
30
90
KN-44 Auto
[local] release orbis BGOYETTE-764 (2341519) Mon Jan 04 11:28:23 2016

Video – Surprising an opponent around a corner.

Call of Duty is all about reacting fast to seeing an opponent.


(-924 -488 464) 0.00 (64 90 0) fov 63
q-aea59e40-US-en
140
100
75
[local] release orbis BGOYETTE-764 (2341513) Mon Jan 04 11:28:23 2016

Video – Line of sight test

One player standing, the other strafing left and right in and out of line of sight.

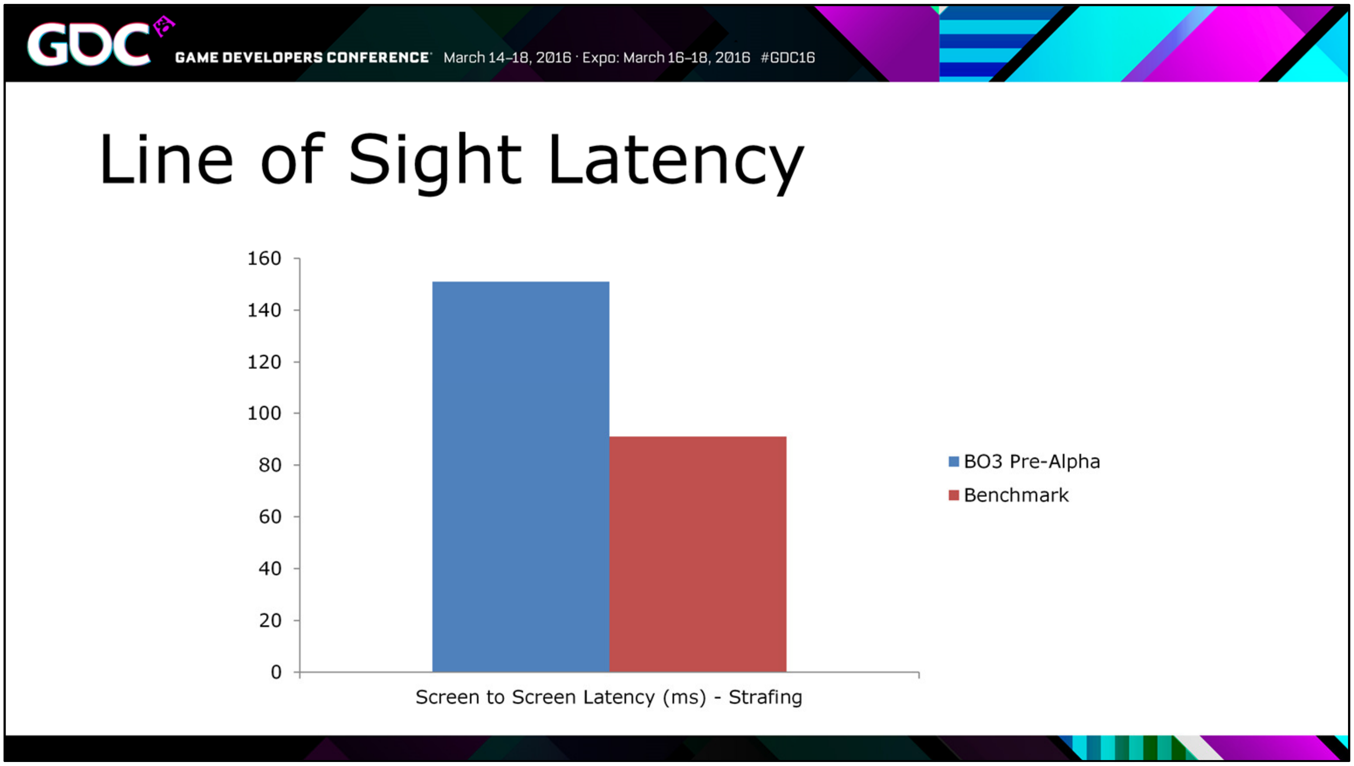
Video – Line of sight test, full speed

Video – Line of sight test, slow motion

Choosing a reference point:
We chose the head as the reference point. Because this is where the camera is located and Call of Duty is about headshots. There’s a high damage modifier on the head and players aim their cross-hair at the height of players head when anticipating an opponent.
Using an elbow or a foot gave irregular measurements.

Again we were expecting to get a good result because of the improvements that were made.

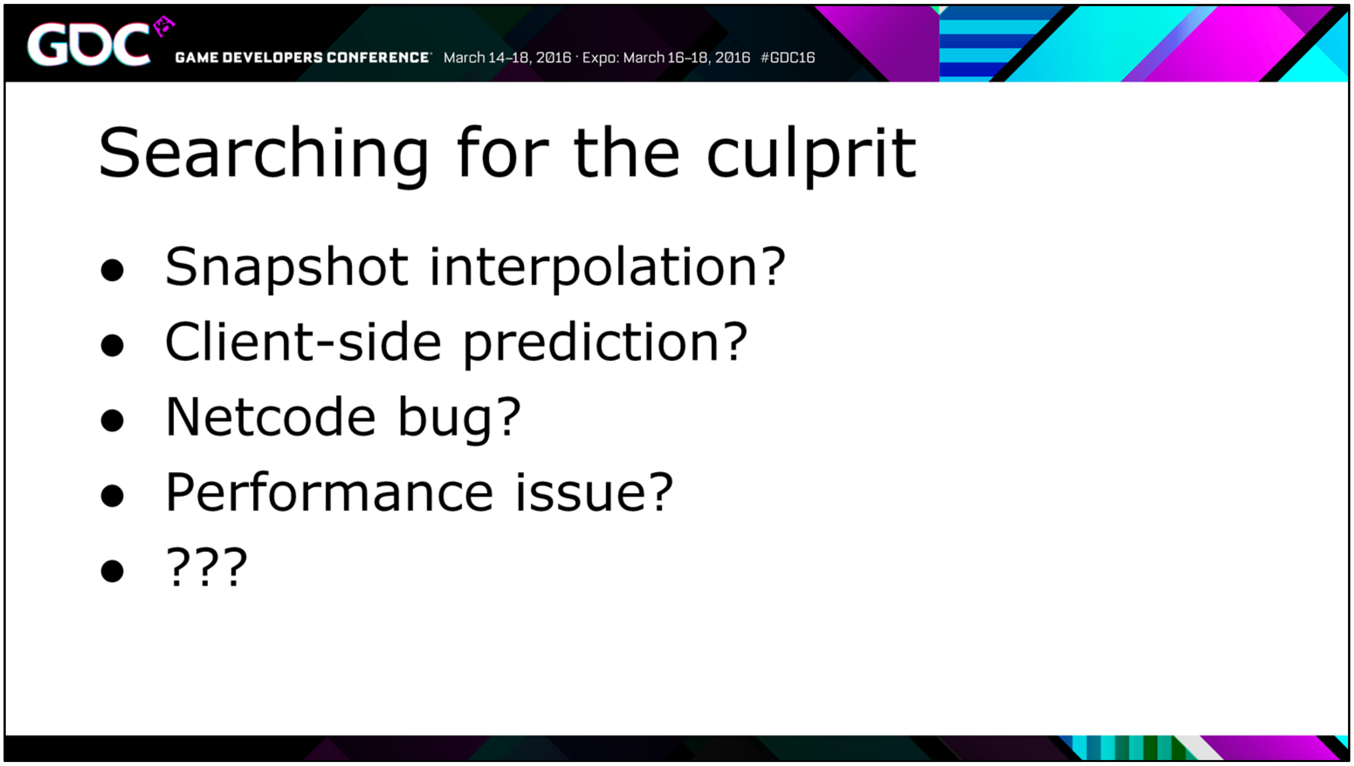
- Increased Snapshot Rate
- Increased Command Rate
- Minimal interpolation
- Script Engine Optimizations
- Snapshot Buffer Improvements



Line of Sight Latency

60ms slower

That was a shock.
But at least we had proof now.
I thought it was going to be easy to fix from there…



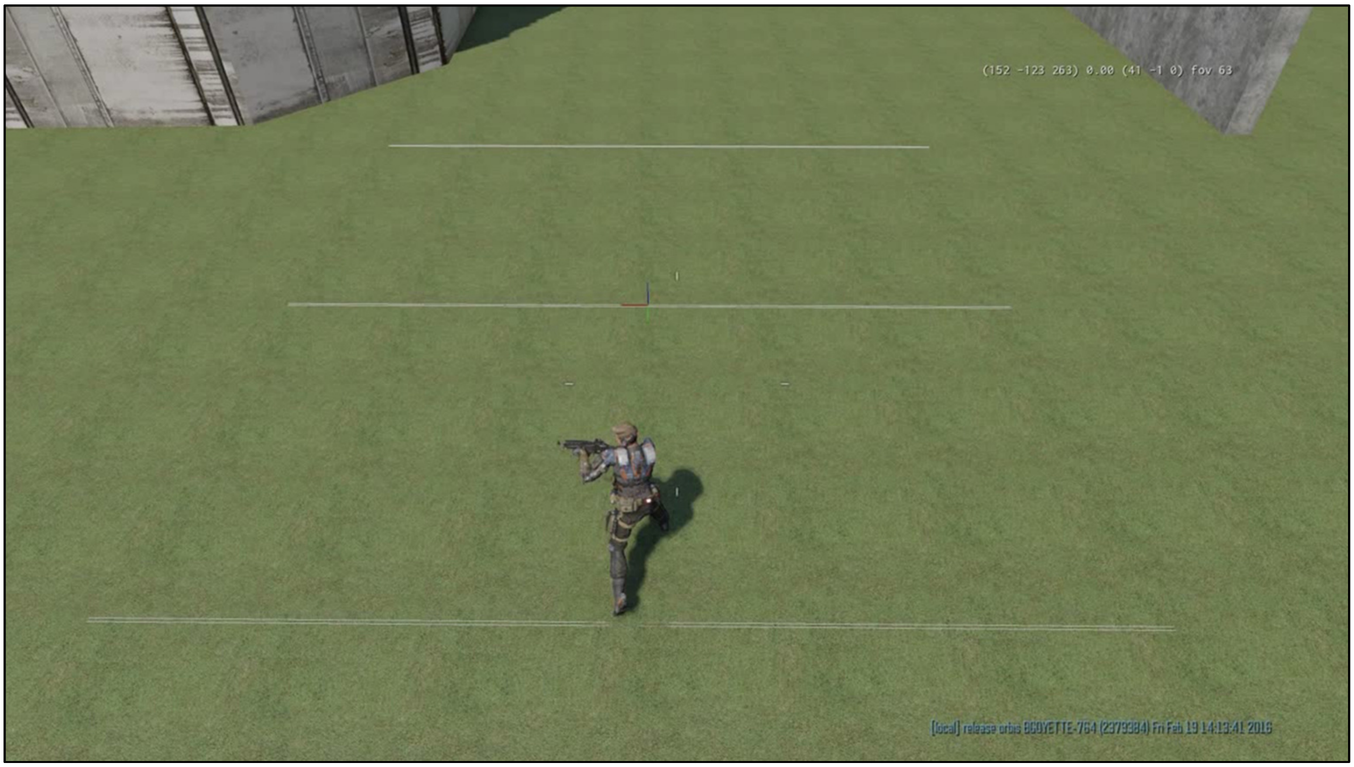
Searching for the culprit

- Snapshot interpolation?
- Client-side prediction?
- Netcode bug?
- Performance issue?
- ???

Searching for a while, I could not find the culprit.

Everywhere I looked, it seemed like the netcode should have been faster than the benchmark.

I audited the whole netcode engine.

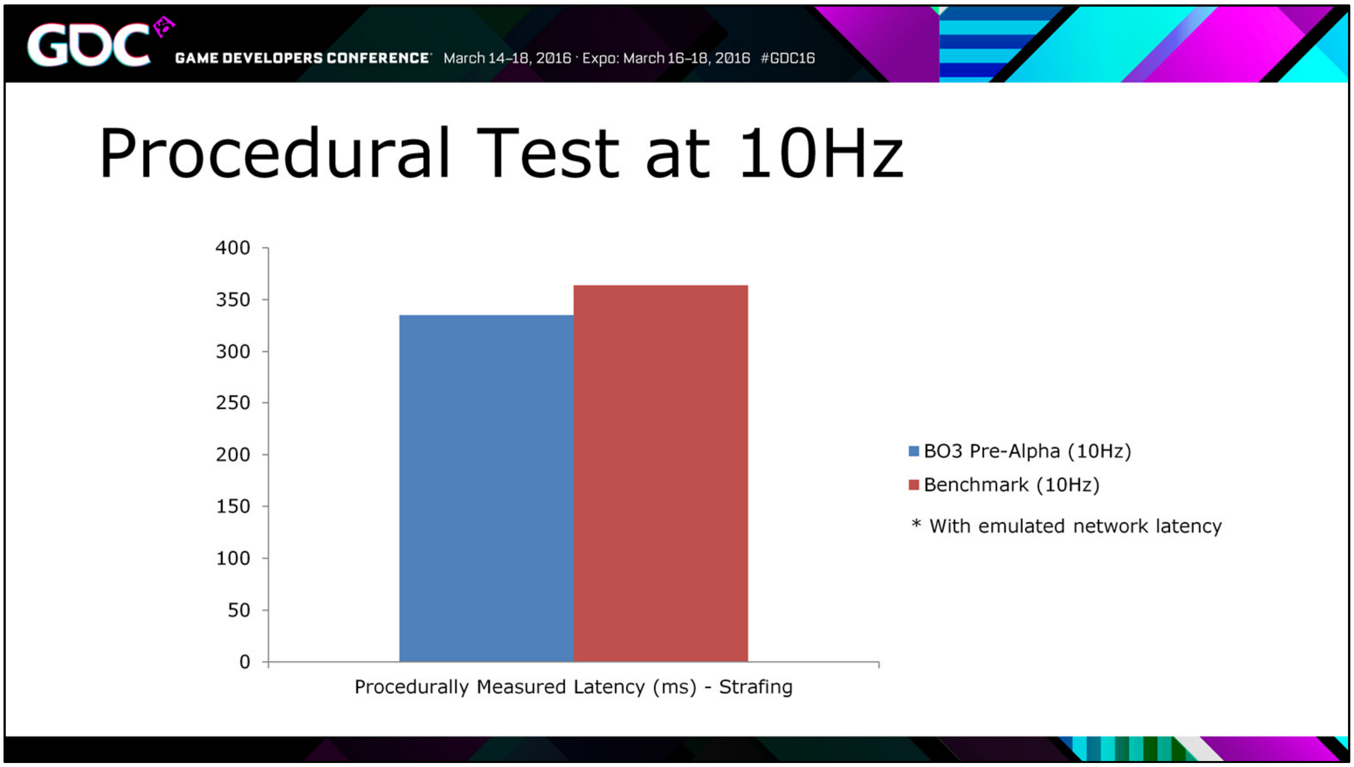

(152 -123 263) 0.00 (41 -1 0) fov 63
[local] release orbis BGOYETTE-764 (2379384) Fri Feb 19 14:13:41 2016

Video – New procedural tool

Getting desperate, I tried to **replicate the camera test** by writing a new procedural tool. The character would strafe left and right over a line. When crossing the line, I would output as much timing data as I could get my hands on. Anything that could potentially affect latency.

Then re-did the test in previous COD and looked at the data to see if I could find a difference…

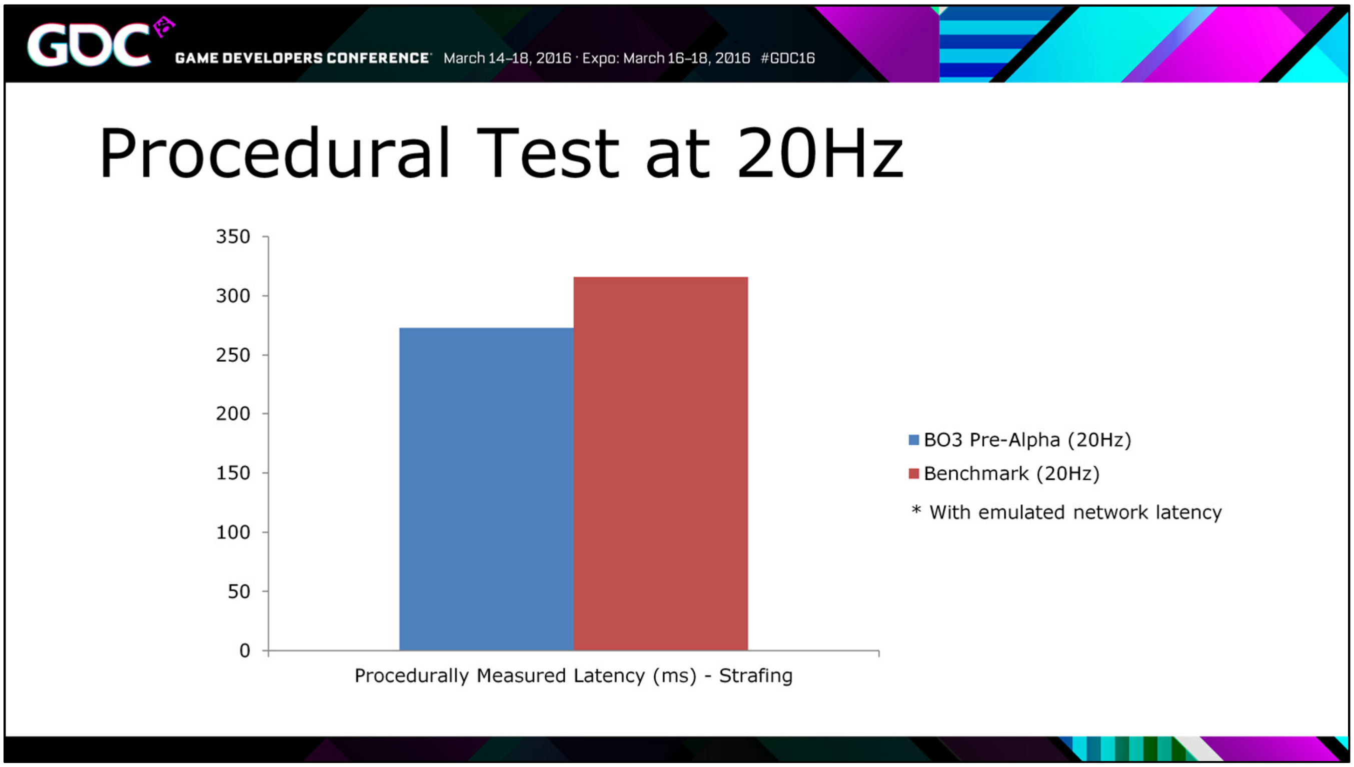
What I found:



GDC GAME DEVELOPERS CONFERENCE March 14–18, 2016 · Expo: March 16–18, 2016 #GDC16

Procedural Test at 10Hz

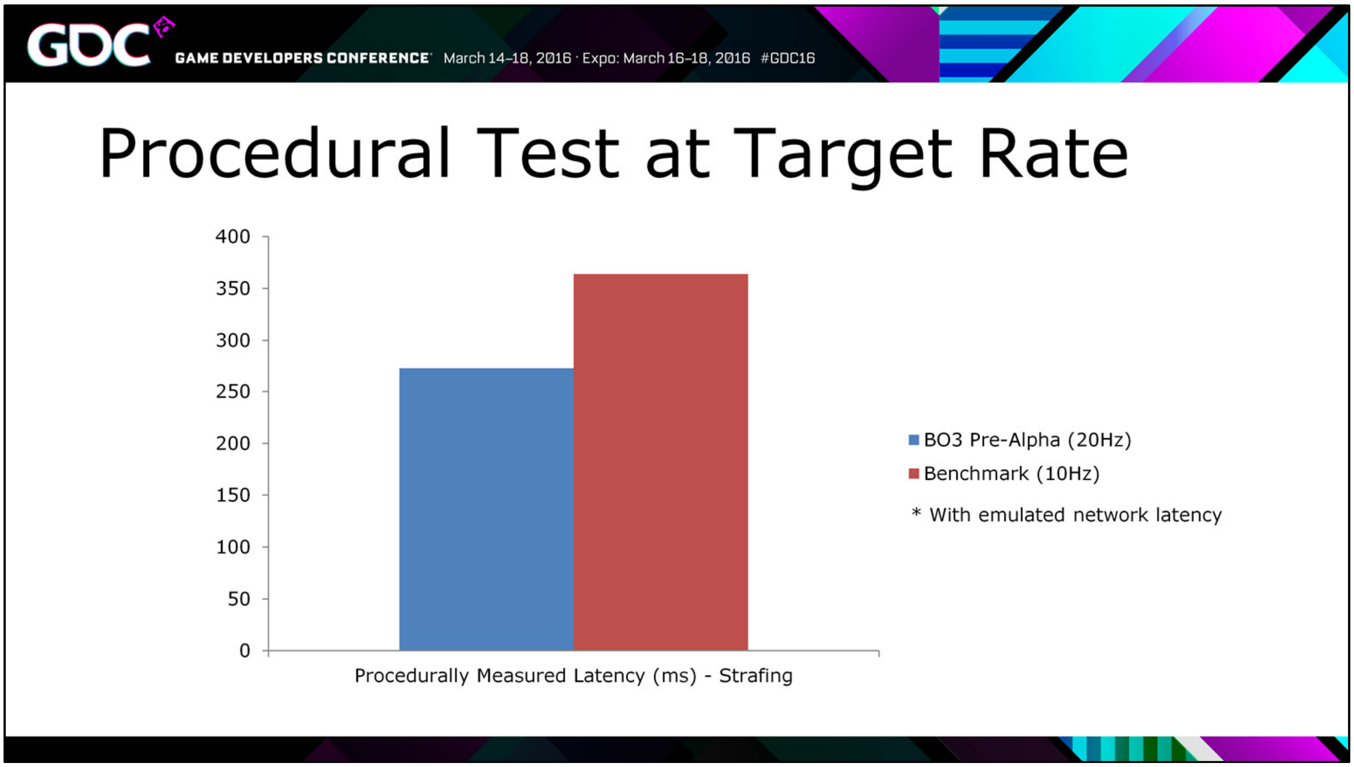
30ms faster at 10Hz



Procedural Test at 20Hz

* With emulated network latency

40ms faster at 20Hz



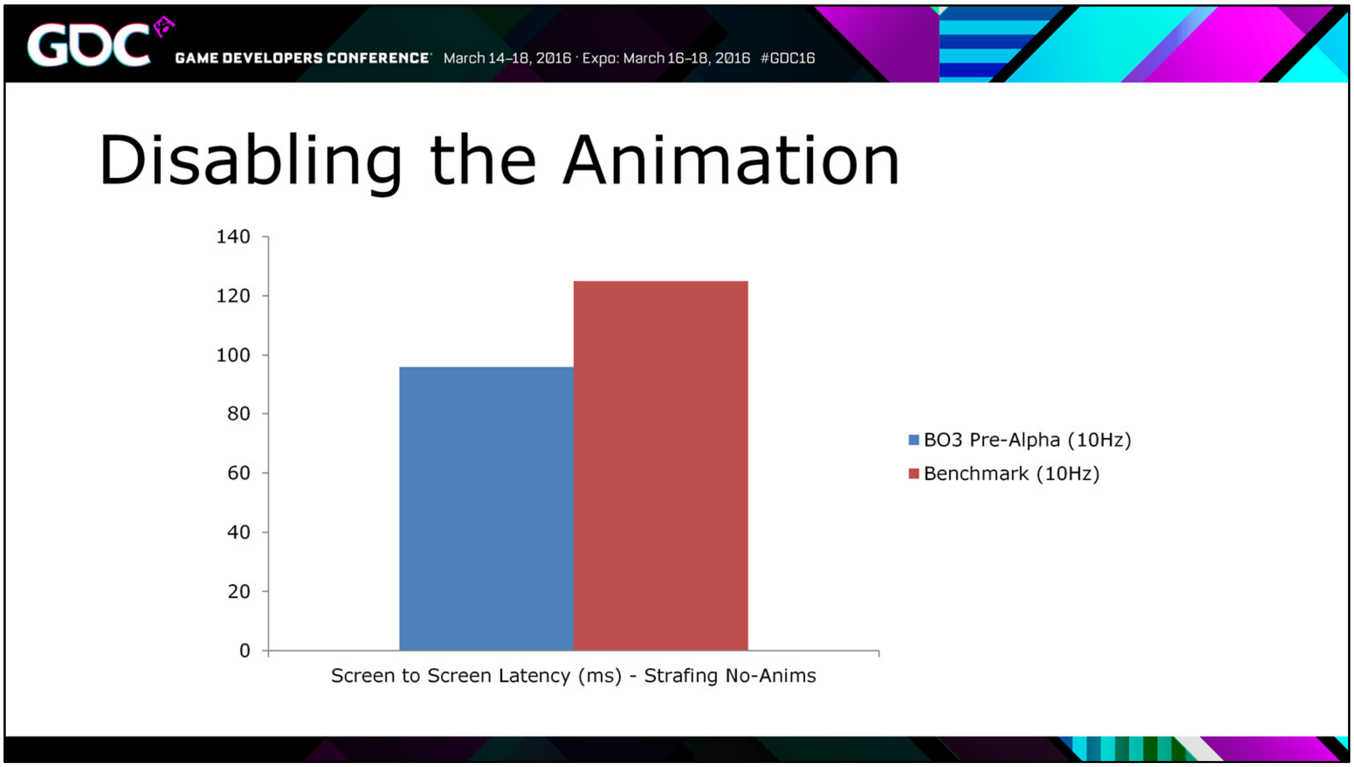
Procedural Test at Target Rate

90 ms faster at their target retail rate.

Ok. So the netcode really was faster. Good. But why was the camera capturing so much latency?

Is it the graphics?
The animation?

I re-did the test by disabling animation – the players were in T-Stance.



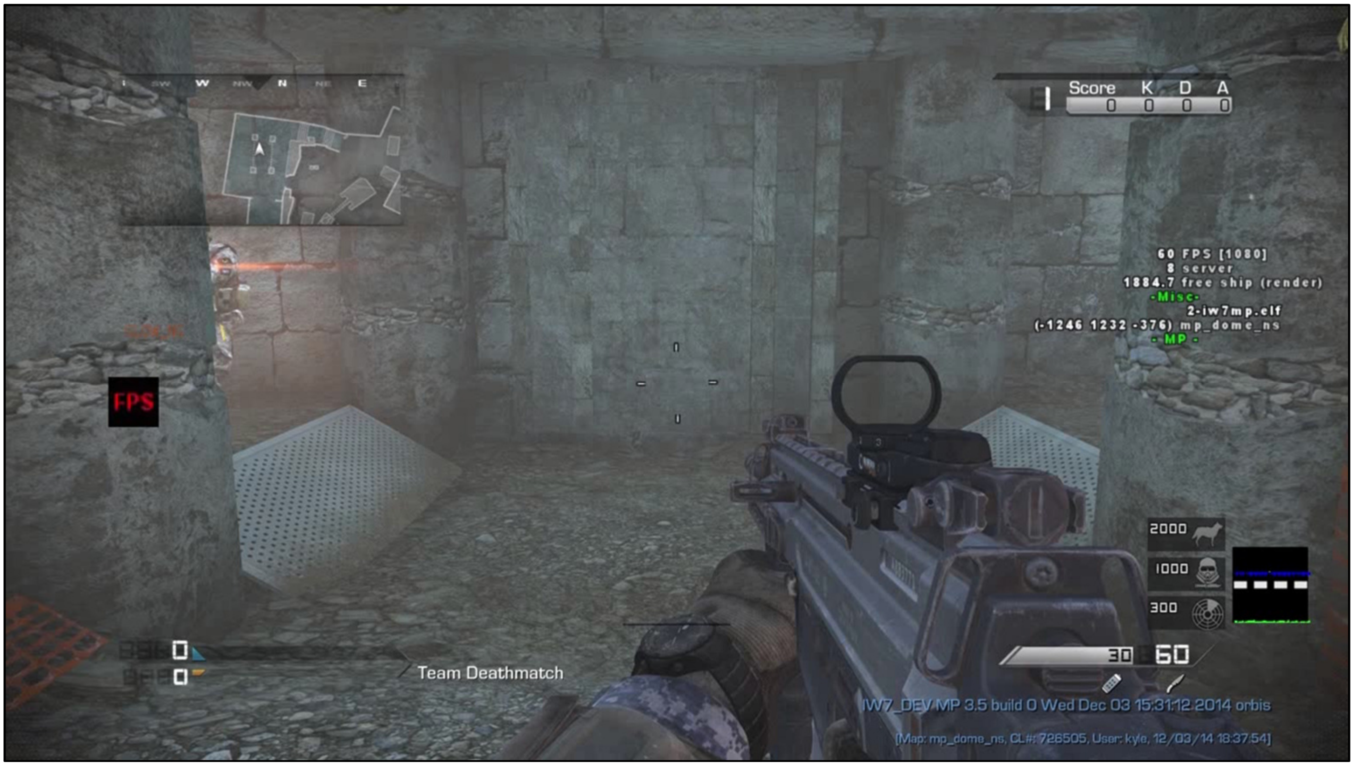
Disabling the Animation

30ms faster

Finally in-line with the camera!
Ok. We have an animation problem.

Though after looking at profiling captures, it wasn’t a problem with the performance of the animation engine…

???



```
60 FPS [1080]
8 server
1884.7 free ship (render)
-Misc-
2-iw7mp.elf
(-1246 1232 -376) mp_dome_ns
- MP -
```

```
IW7_DEV MP 3.5 build 0 Wed Dec 03 15:31:12 2014 orbis
[Map: mp_dome_ns, CL#: 726505, User: kyle, 12/03/14 18:37:54]
```

Video – Looking at the animation style of Call of Duty Ghosts when strafing.

The player **leans** in the direction he's strafing.
The lean is **symmetrical** in both directions.
This puts the head slightly ahead of the entity marker (the position synced in the snapshots)

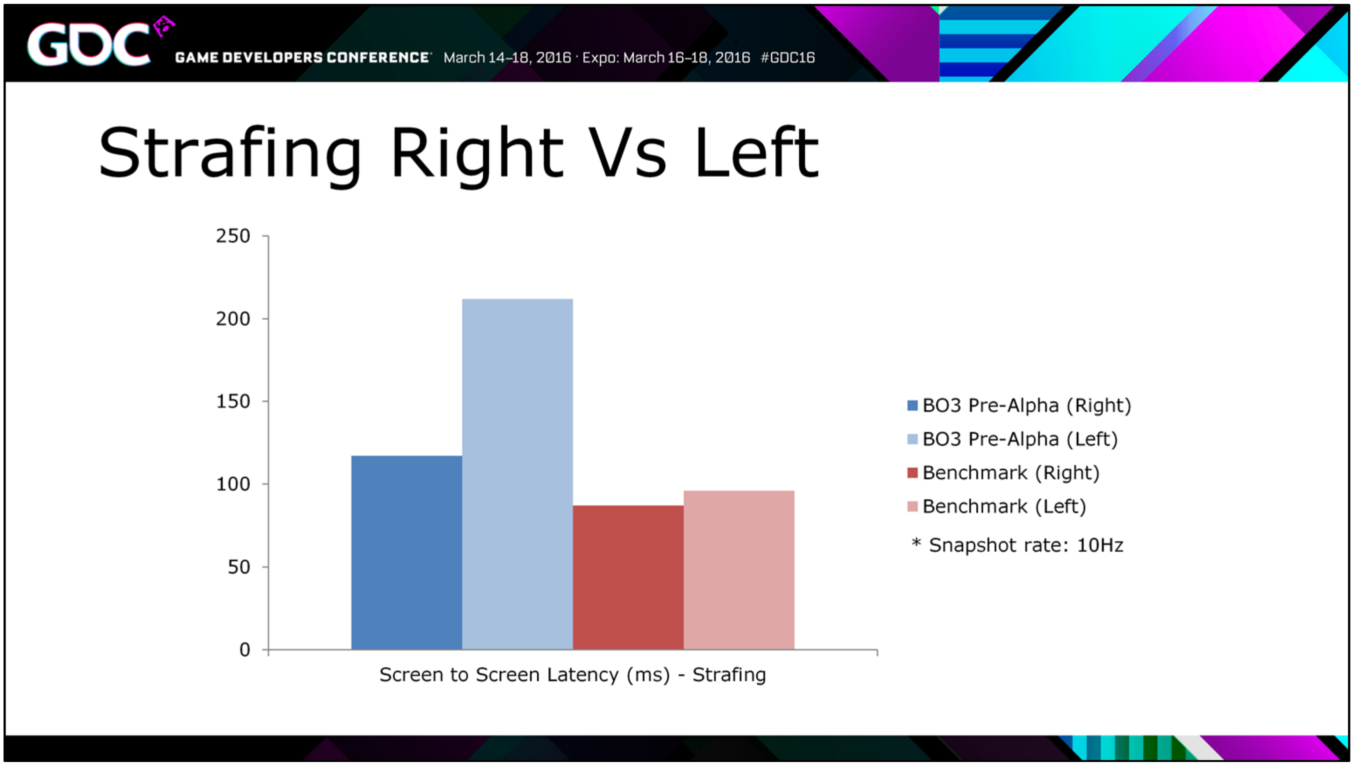
(-901 -164 60) 0.00 (8 -90 0) fov 63
[local] release orbis BGOYETTE-764 (2341513) Mon Jan 04 11:28:23 2016

Video – Animation style of Black Ops 3 when strafing.

The animation style is different.
The players leans in a similar way when strafing right, but walks backward when strafing left.
This puts the head ahead of the entity marker when strafing right, but the head stays behind the marker when strafing left.

Could this small detail have an impact on perceived latency?

Testing again but this time looking at left / right separately (my previous tests did not make that distinction).



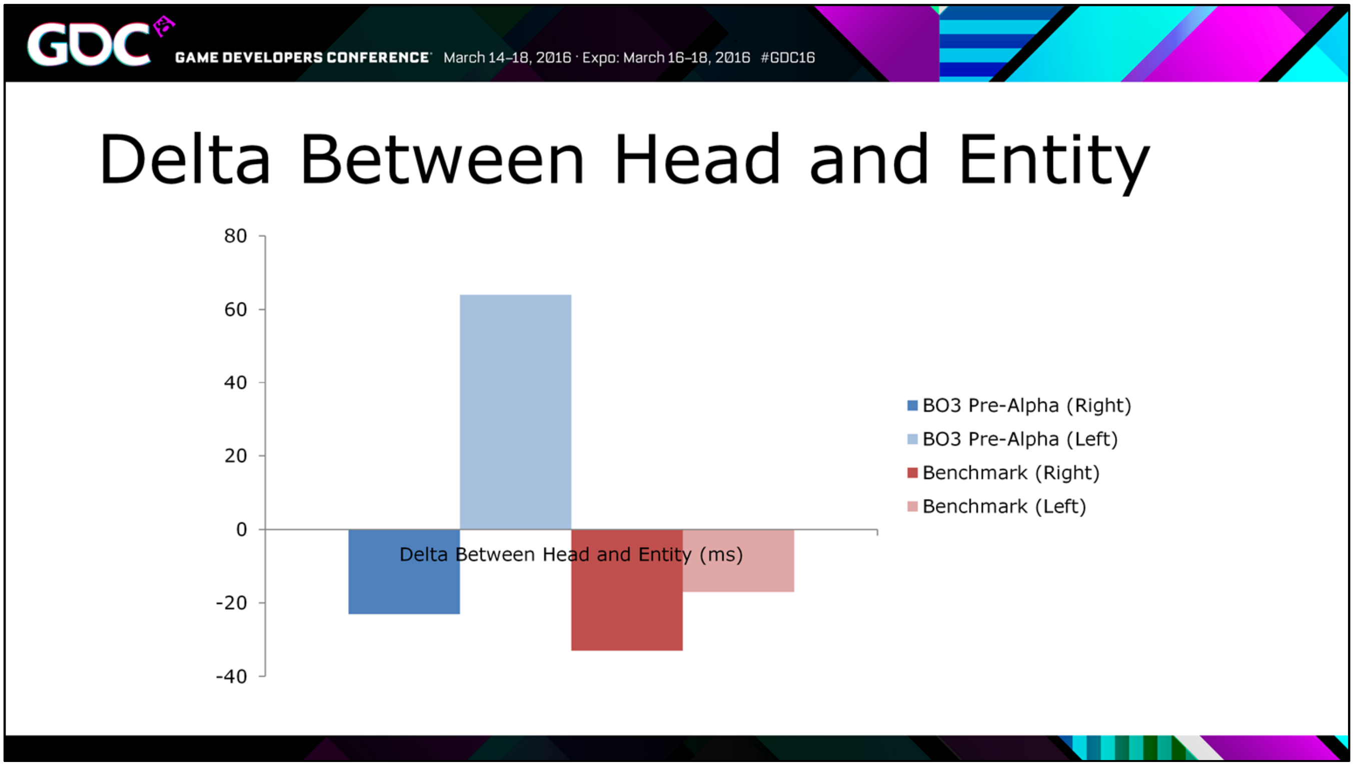
Strafing Right Vs Left

* Snapshot rate: 10Hz

100ms slower when strafing left!!

This was with a camera test with animation.

Next, looking at just the gap between the head and the entity marker.



Delta Between Head and Entity

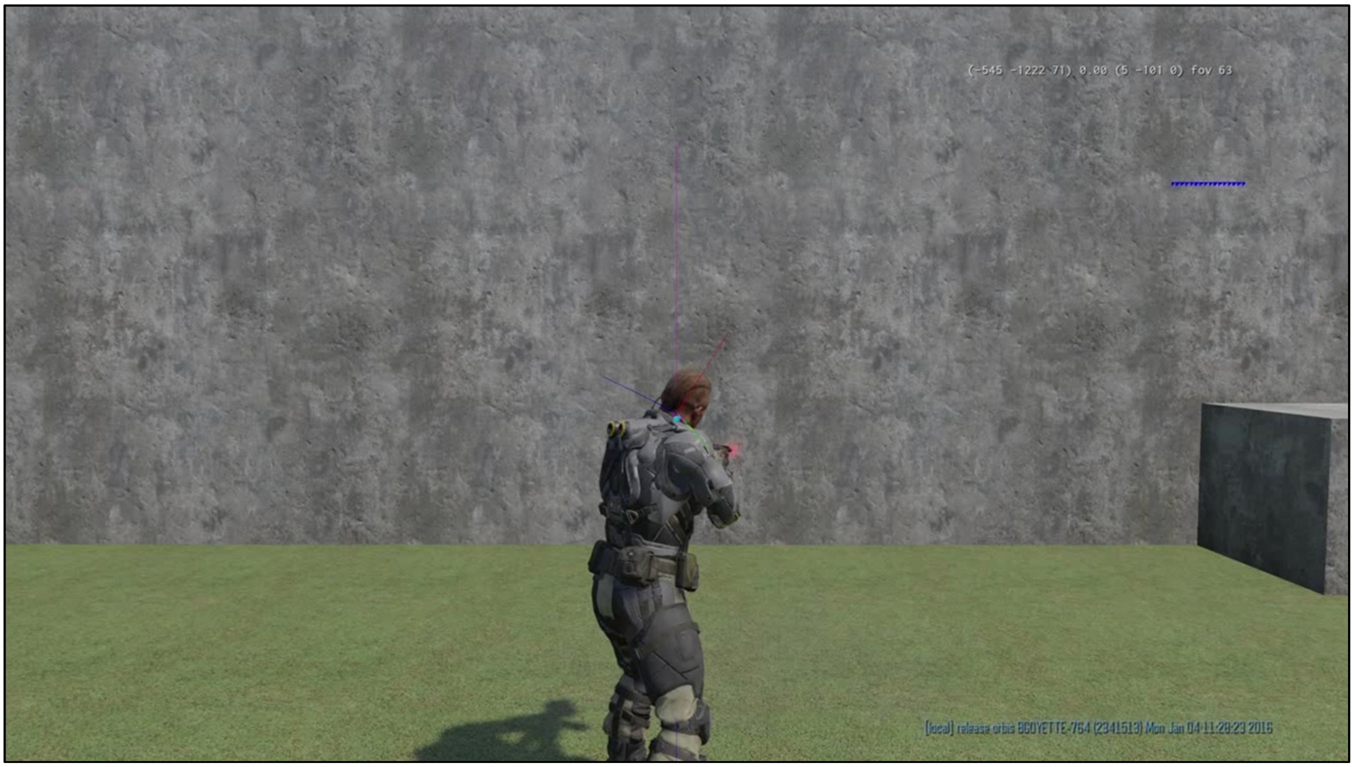
87ms slower when strafing left!!
Negative latency otherwise.

We can actually beat the network latency by a non-negligible amount by having the head slightly ahead of the snapshot position.

What if we would use that on purpose to our advantage?

- Draw the head model where the player would be looking at us from.
- Only for strafing though, since that’s when players can see each other while moving.
- Hit detection is based on the bones position, so it won’t mess with the accuracy of hit detection.

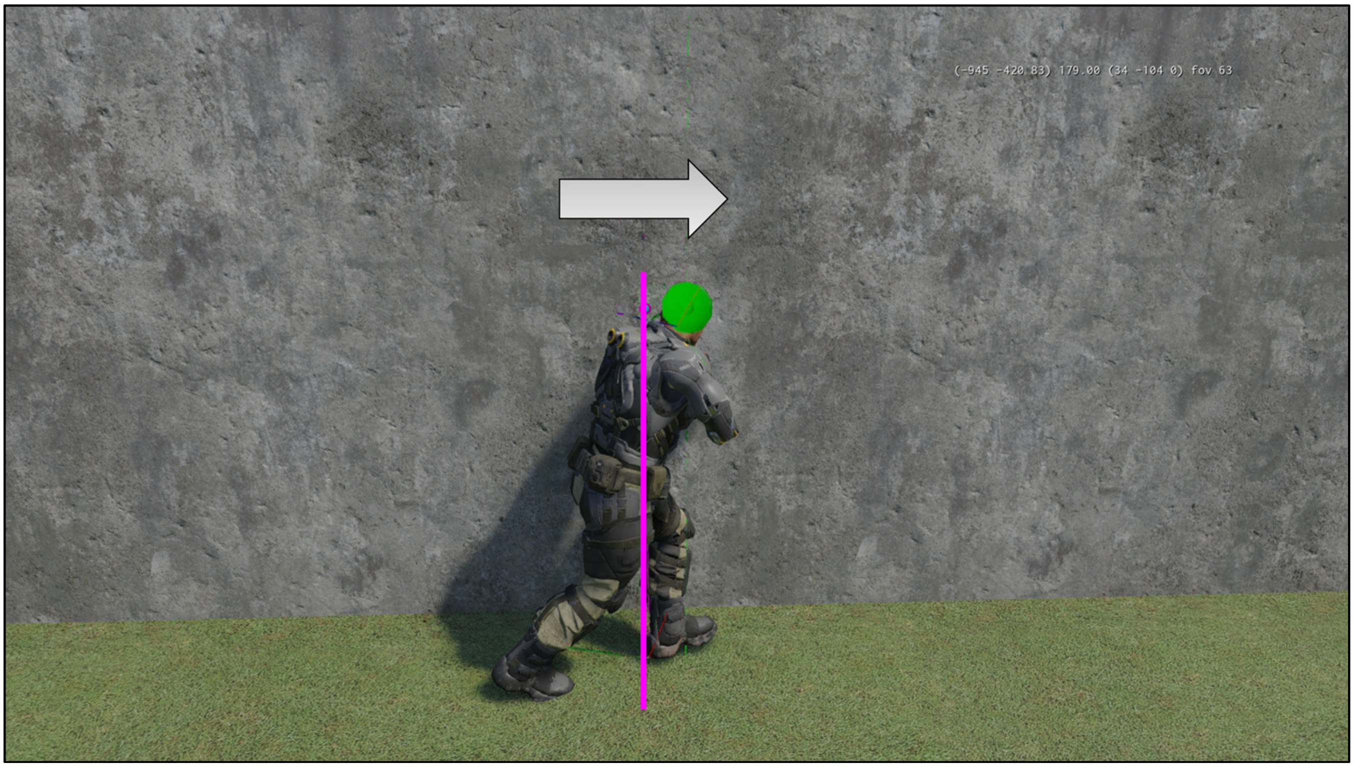
The competitive advantage of seeing the opponent’s head first is huge. Before, the attacking player could headshot his opponent before he could even see the head of his attacker.



```
(-545 -1222 71) 0.00 (5 -101 0) fov 63
[local] release orbis BGOYETTE-764 (2341513) Mon Jan 04 11:28:23 2016
```

Video – New tool for animator to recalibrate animations.

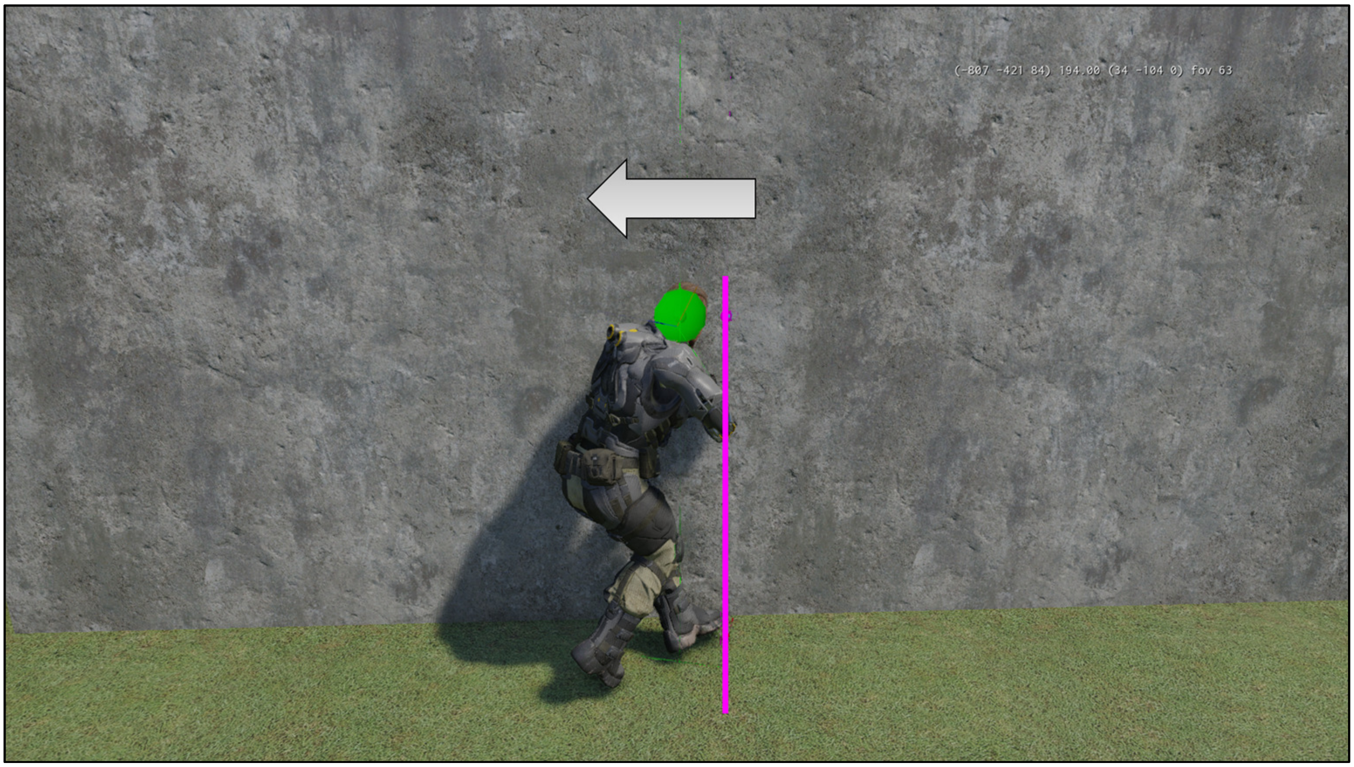
Hiding about 35ms of latency. We reach a limit where it wouldn’t look good to try to hide more.



Draw a sphere where we want the head to be.

The purple line is the snapshot position (the entity position).
The sphere’s distance from the line is relative to the velocity of the player. The faster the farther.
The sphere is green when the head bone is within range.
The sphere turns red otherwise.

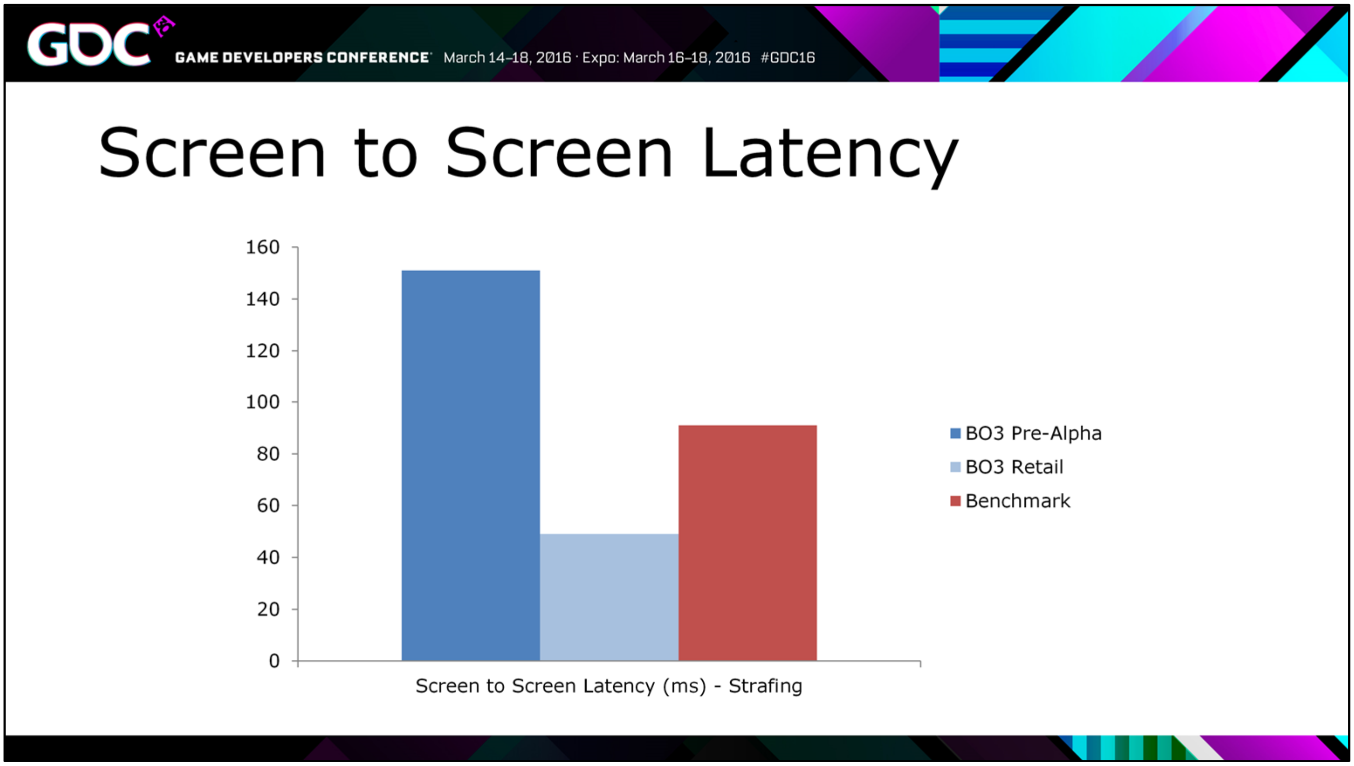
This guides animators.



```
(-807 -421 84) 194.00 (34 -104 0) fov 63
```

Strafing left was changed the most.

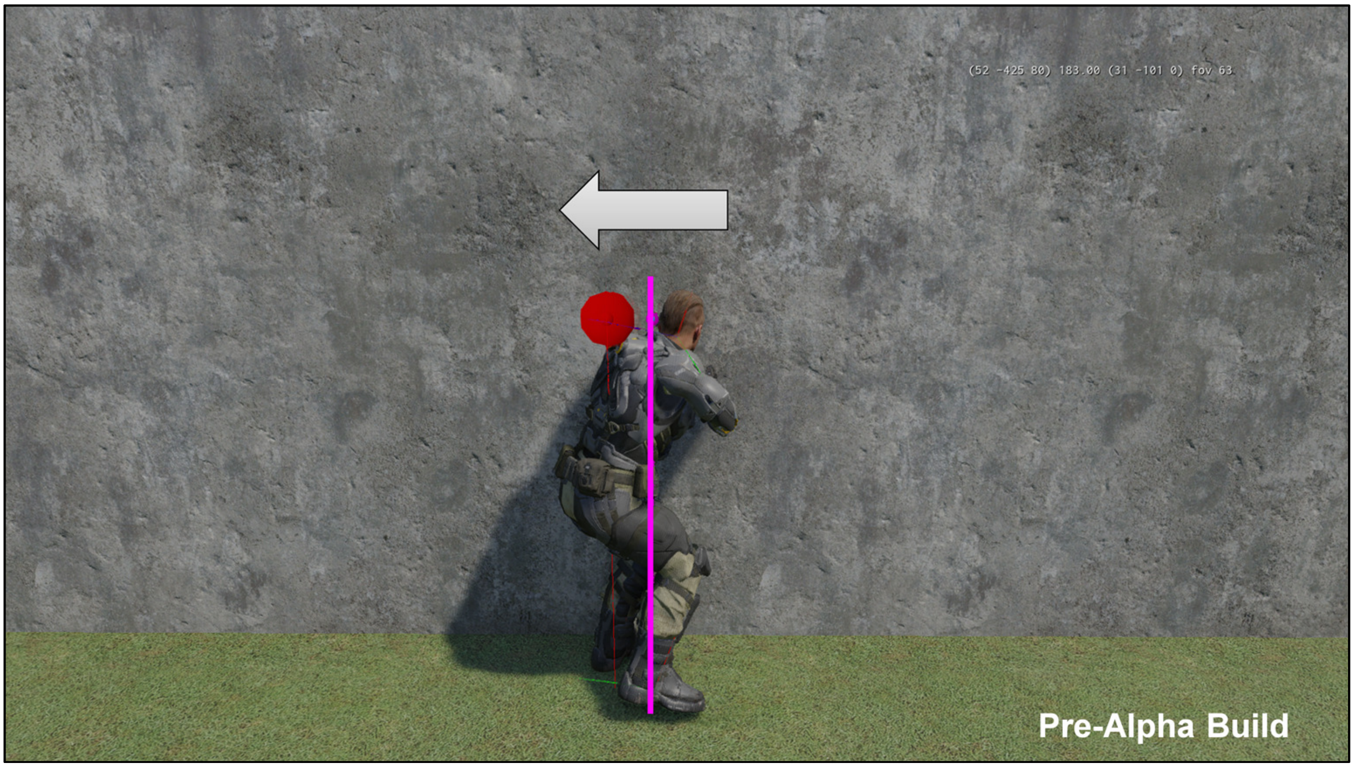
Seeing other body parts sticking out (ass, elbow, foot, tip of the gun, etc) is part of the game (and should be considered by designers).
But the head is the most critical body part to get right.



Screen to Screen Latency

After the changes.

102 ms faster than previous builds
42ms faster than benchmark



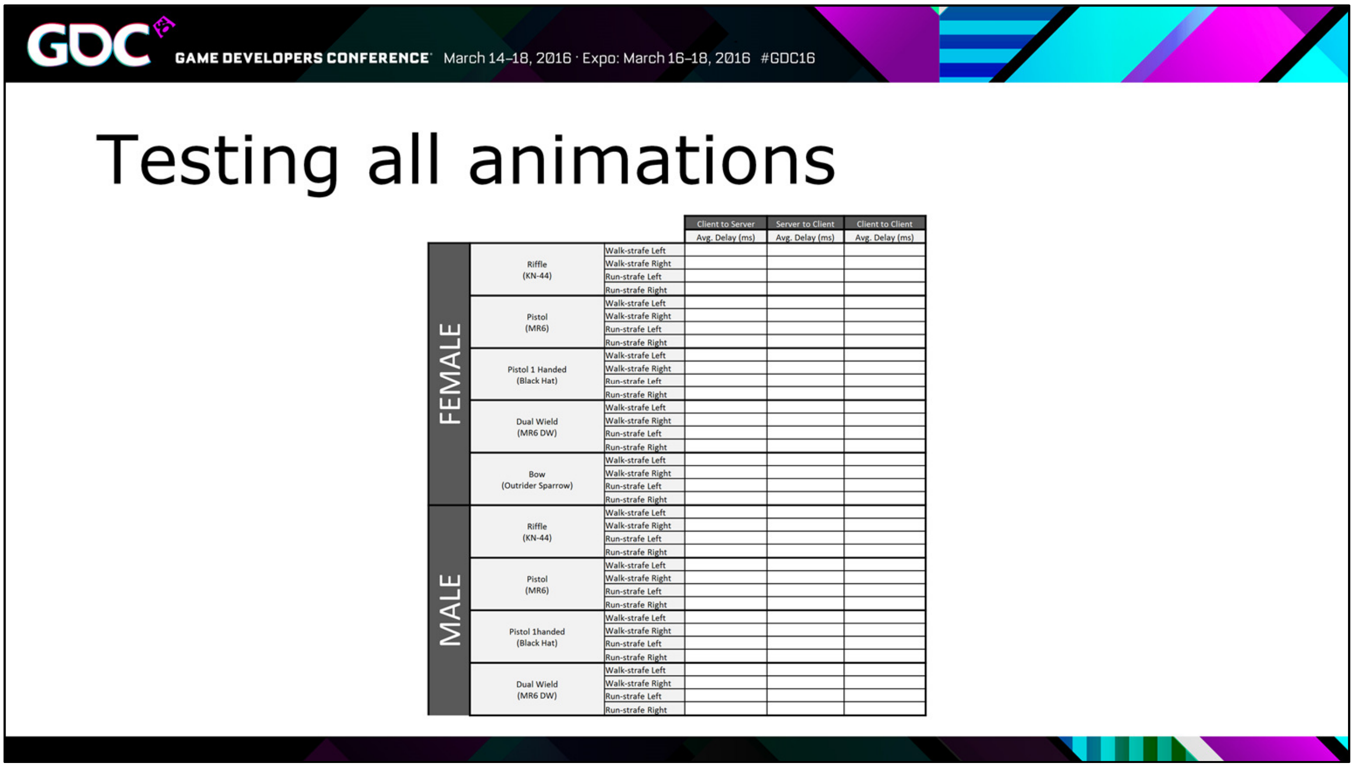
The original strafing left animation was off only by this much.

But it was worth 100ms

ONE HUNDRED MILLISECONDS

As a network programmer I can write netcode all day trying to optimize everything I do. And if I was ever able to reduce everyone’s ping by 30ms, it would be a huge accomplishment. This was worth 100.

It was a big surprise.
I was not expecting the animation to have such a big impact on perceived latency.



Testing all animations

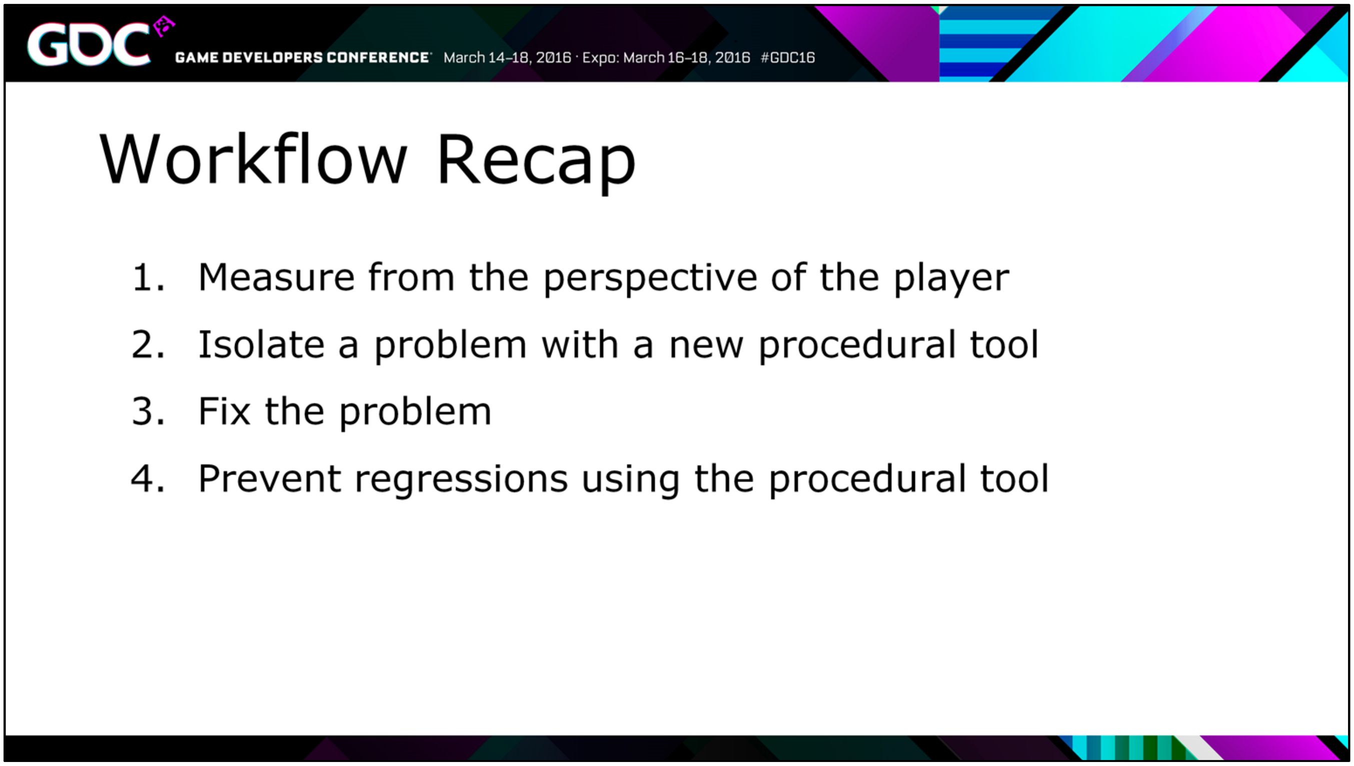
			Client to Server	Server to Client	Client to Client
			Avg. Delay (ms)	Avg. Delay (ms)	Avg. Delay (ms)
FEMALE	Riffle (KN-44)	Walk-strafe Left			
		Walk-strafe Right			
		Run-strafe Left			
		Run-strafe Right			
	Pistol (MR6)	Walk-strafe Left			
		Walk-strafe Right			
		Run-strafe Left			
		Run-strafe Right			
	Pistol 1 Handed (Black Hat)	Walk-strafe Left			
		Walk-strafe Right			
		Run-strafe Left			
		Run-strafe Right			
	Dual Wield (MR6 DW)	Walk-strafe Left			
		Walk-strafe Right			
		Run-strafe Left			
		Run-strafe Right			
	Bow (Outrider Sparrow)	Walk-strafe Left			
		Walk-strafe Right			
		Run-strafe Left			
		Run-strafe Right			
MALE	Riffle (KN-44)	Walk-strafe Left			
		Walk-strafe Right			
		Run-strafe Left			
		Run-strafe Right			
	Pistol (MR6)	Walk-strafe Left			
		Walk-strafe Right			
		Run-strafe Left			
		Run-strafe Right			
	Pistol 1handed (Black Hat)	Walk-strafe Left			
		Walk-strafe Right			
		Run-strafe Left			
		Run-strafe Right			
	Dual Wield (MR6 DW)	Walk-strafe Left			
		Walk-strafe Right			
		Run-strafe Left			
		Run-strafe Right			

Testing all animation combinations.

THANK GOD FOR QA

The procedural tool was updated to consider the head bone instead of the entity position.
It also measured the left and right strafe separately.

We found problems with many anims and we were able to fix them.



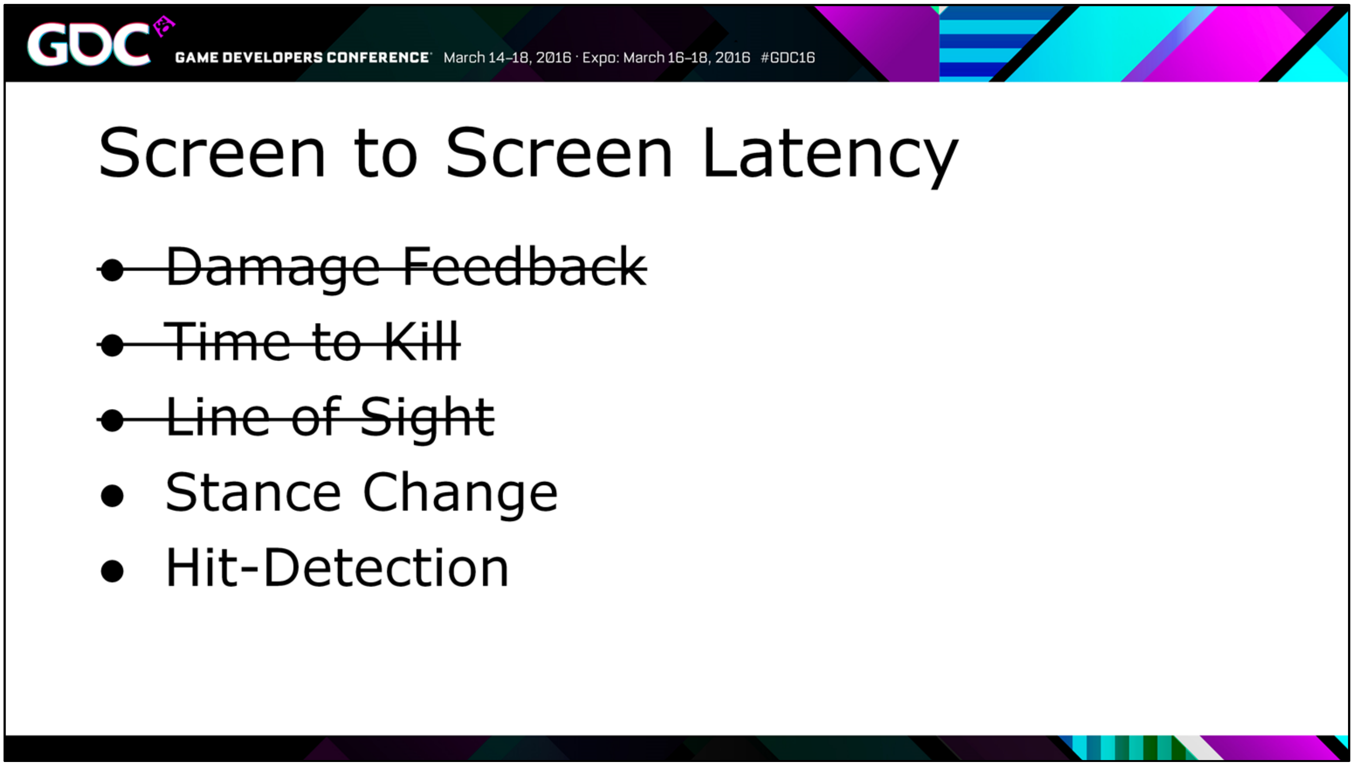
Workflow Recap

1. Measure from the perspective of the player
2. Isolate a problem with a new procedural tool
3. Fix the problem
4. Prevent regressions using the procedural tool

The idea isn’t to use the camera every day. But to use it to get a new perspective and validate your procedural tools.

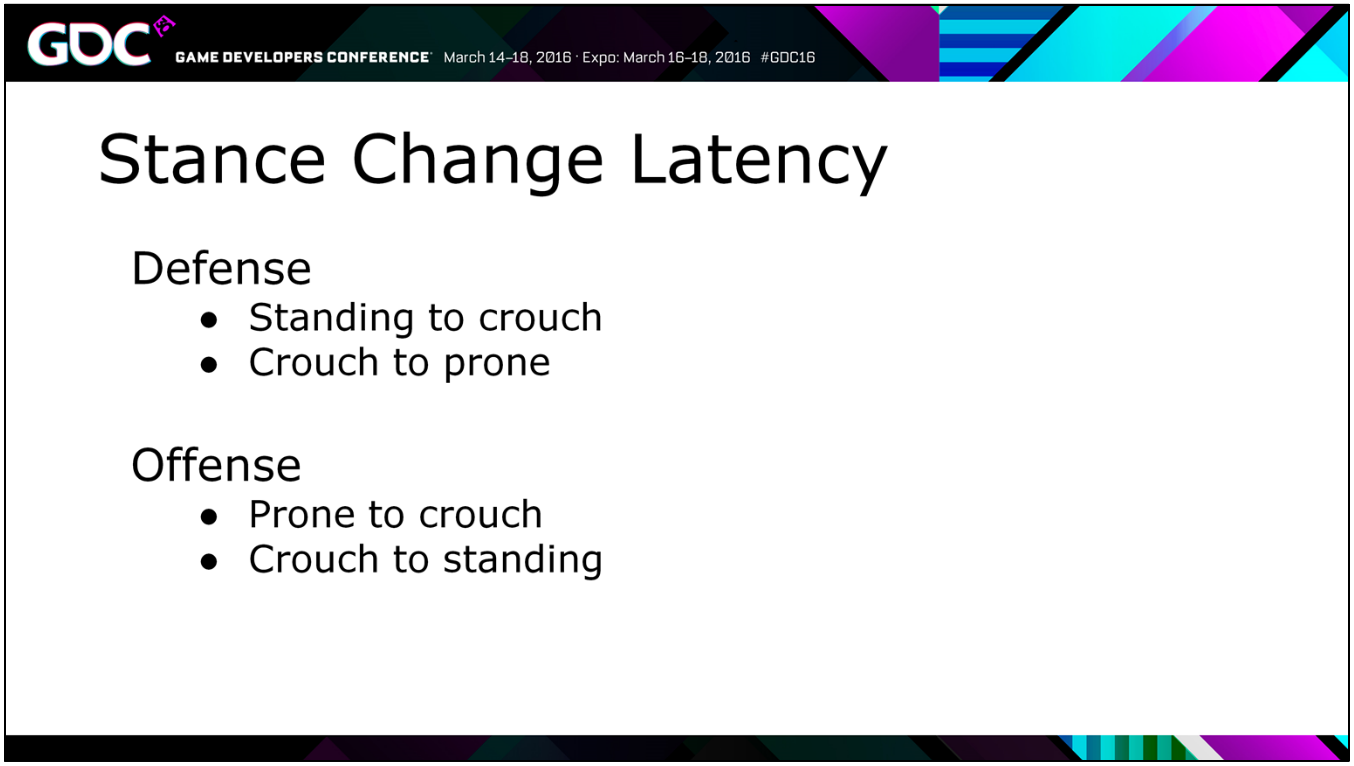
Then most of the time you iterate with your procedural tools.

And once you have the camera setup, it’s easy to just record a quick test to validate an assumption.



Screen to Screen Latency

- ~~Damage Feedback~~
- ~~Time to Kill~~
- ~~Line of Sight~~
- Stance Change
- Hit-Detection



Stance Change Latency

Defense

- Standing to crouch
- Crouch to prone

Offense

- Prone to crouch
- Crouch to standing

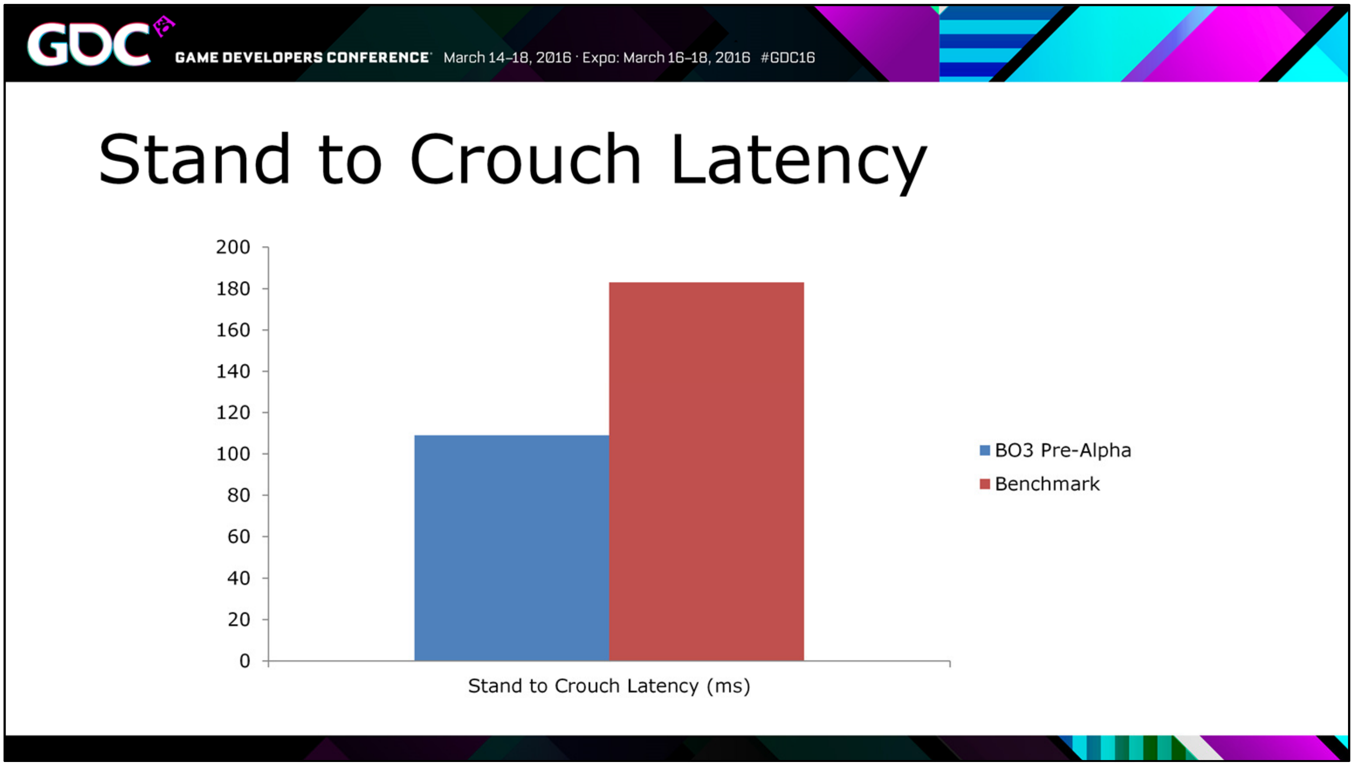
Why is it important?

A discrepancy in the 1st person and 3rd person animation can give a competitive advantage to one player. You can either see the other player earlier or they can still see you after you took cover.

Video – Stance change latency test, full speed

Video – Stance change latency test, slow motion

There's more latency when going down because of client-side prediction; So that it feels responsive.
But when going up, we can cheat with a slower 1st person anim, giving time for the 3rd person anim to catch up.
So they'll see each other about at the same time.

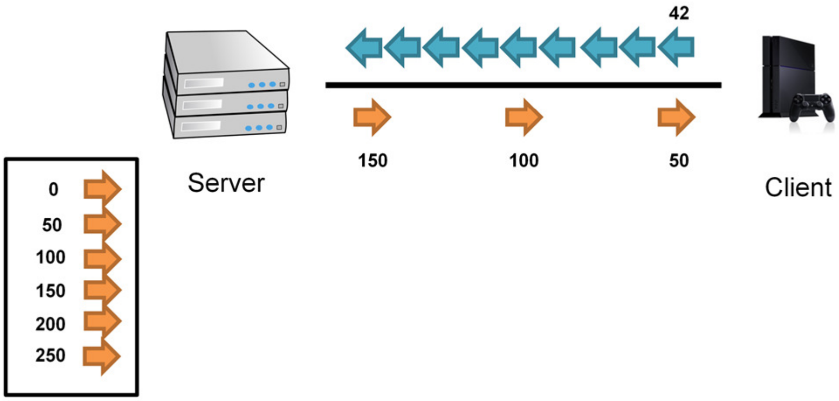


Stand to Crouch Latency

74ms faster

We found the prone to standing animation was off and fixed it by slowing down the 1st person anim.

Hit-Detection Fidelity

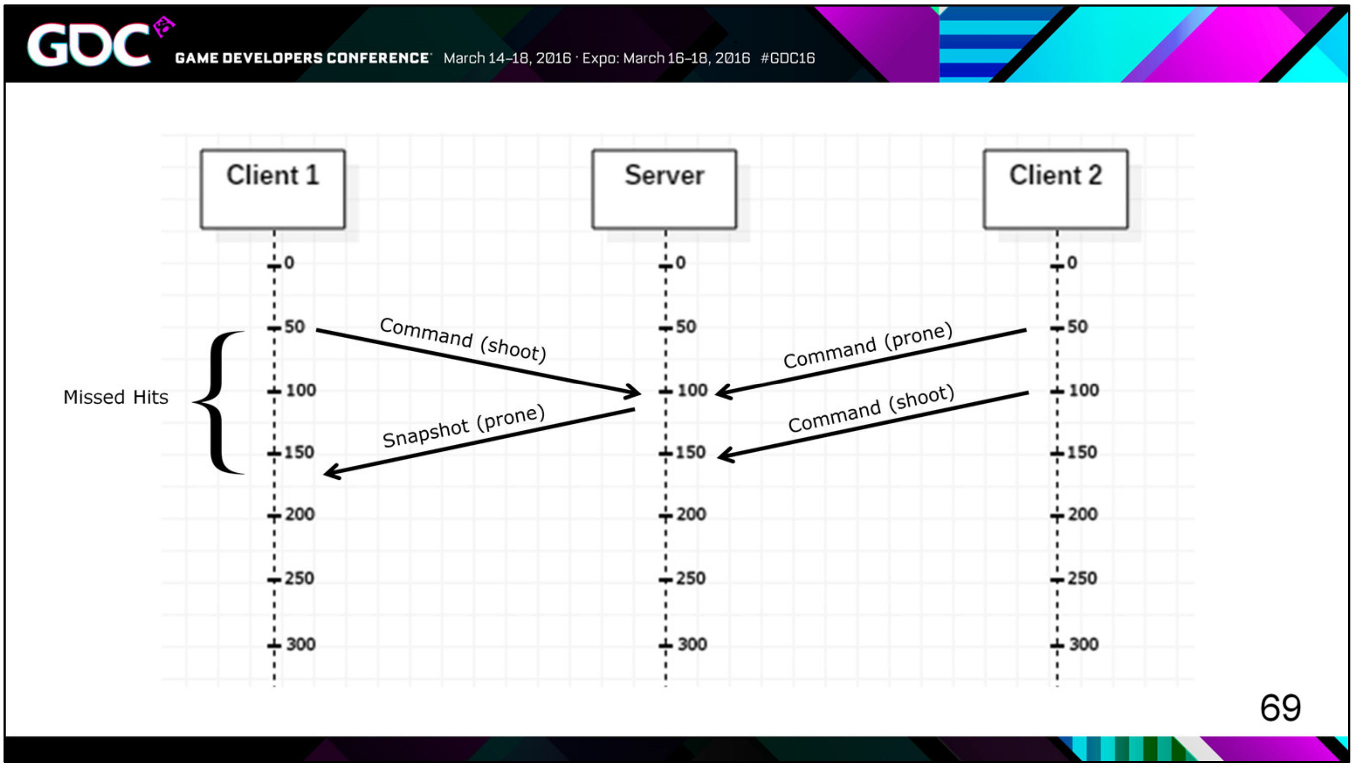


Verifying the Snapshot Buffer, for Hit-Detection Fidelity.

Why is it important?
Because an imprecision in the hit-detection would feel like network latency.

The snapshot buffer originally did not store the animation state; Only the player position & orientation.
When processing hit-detection, the server would use the “present time” animation state.

So if a player changed state, the hits could miss.



GDC GAME DEVELOPERS CONFERENCE March 14–18, 2016 · Expo: March 16–18, 2016 #GDC16

Client 1

Server

Client 2

Missed Hits

Command (shoot)

Command (prone)

Snapshot (prone)

Command (shoot)

UML Sequence Diagram of the Drop-Shooting technique: Changing stance just before shooting.

There’s a window of missed hits there. The higher the ping of the victim, the wider the window.

Treyarch wanted to fix this. The animation state was added to the snapshot buffer. It was very expensive, but was worth it.

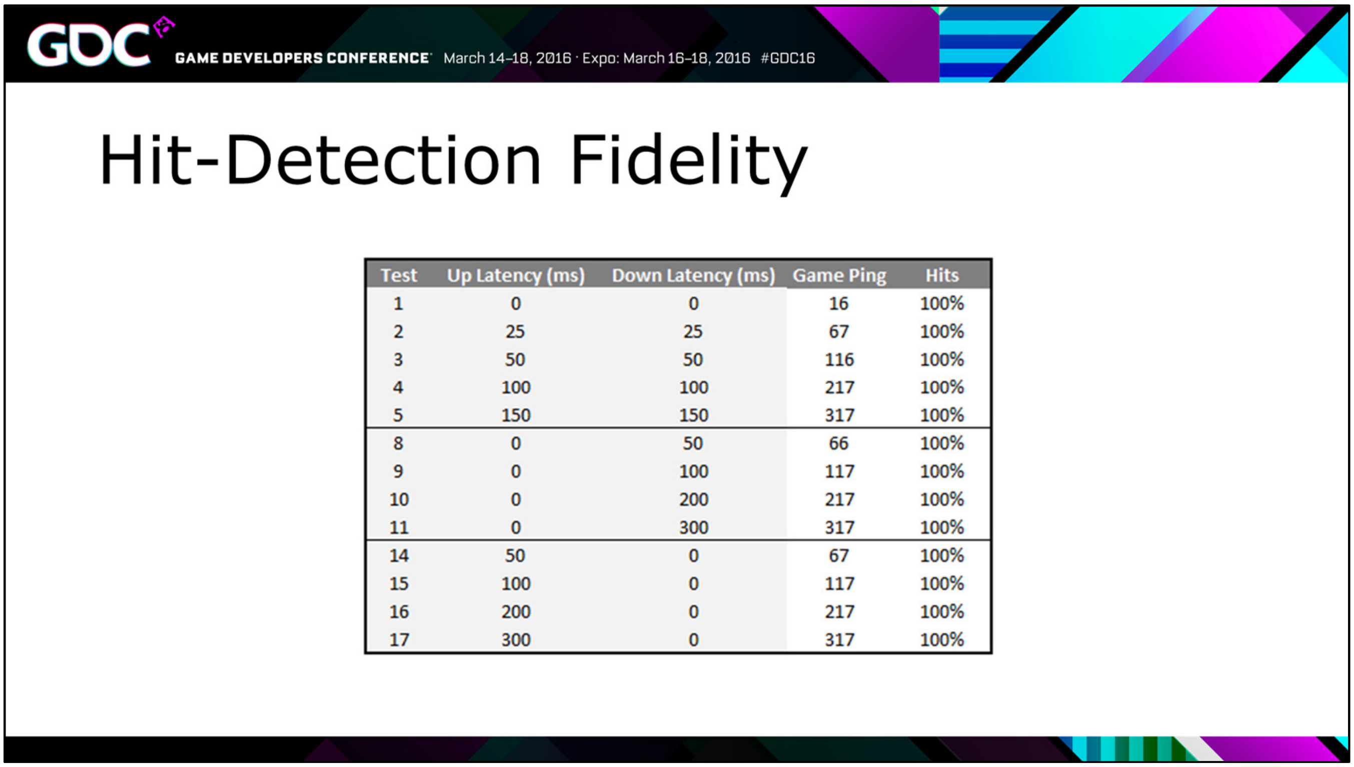
Changing stance is still a legit gameplay technique; You’re harder to shoot at.
But now the effectiveness can be tweaked by game designers and is the same all the time.
It is not affected by network conditions anymore.

Video – Snapshot Buffer Test, No Animation State, Missed Hits

For this test I simply changed the code to fire automatically on the same frame as a stance change.
Without the animation state in the snapshot buffer, the hits miss 100% with just a little bit of emulated network latency.

Video – Snapshot Buffer Test, With Animation State, Hits Register

Now the test is passing.

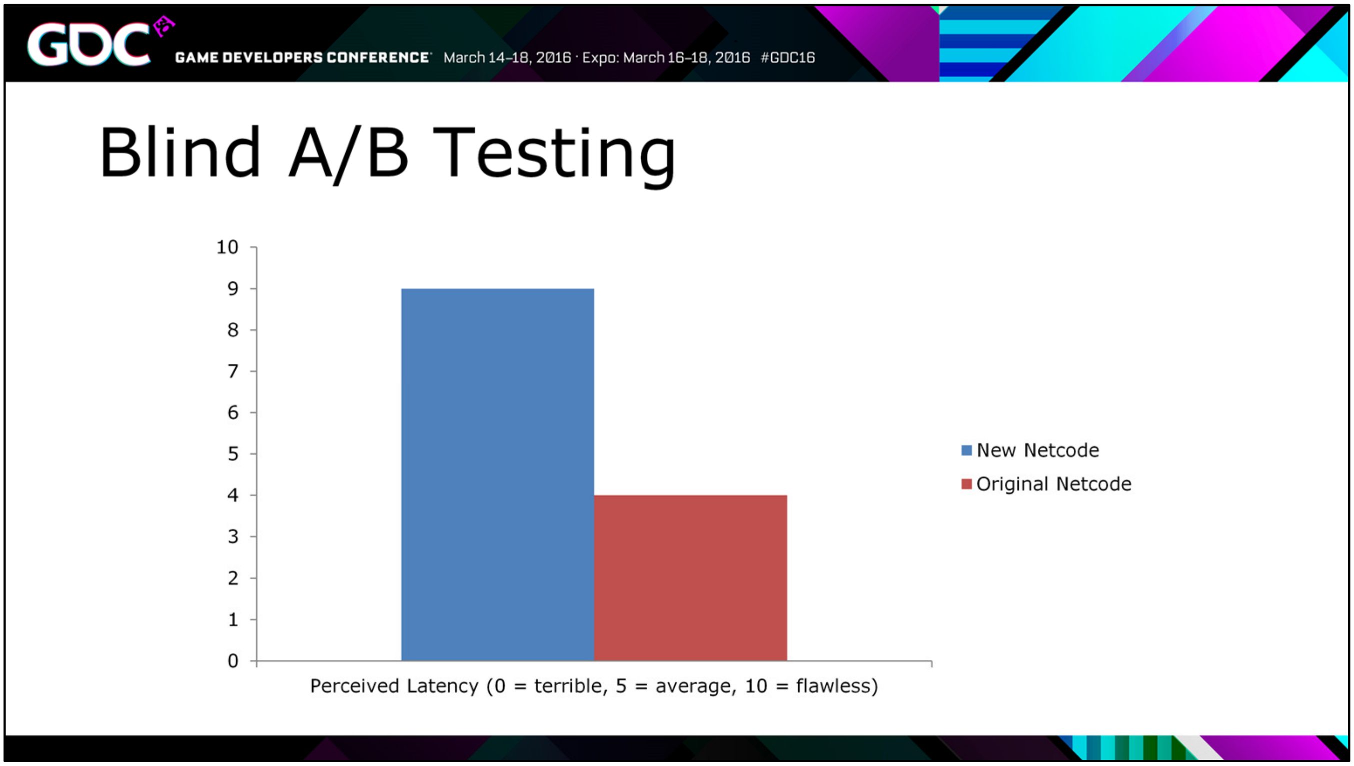


Hit-Detection Fidelity

Test	Up Latency (ms)	Down Latency (ms)	Game Ping	Hits
1	0	0	16	100%
2	25	25	67	100%
3	50	50	116	100%
4	100	100	217	100%
5	150	150	317	100%
8	0	50	66	100%
9	0	100	117	100%
10	0	200	217	100%
11	0	300	317	100%
14	50	0	67	100%
15	100	0	117	100%
16	200	0	217	100%
17	300	0	317	100%

The Rewind Buffer worked 100% of the time in the various network conditions tested.

The asymmetry of the ping don't impact the result.



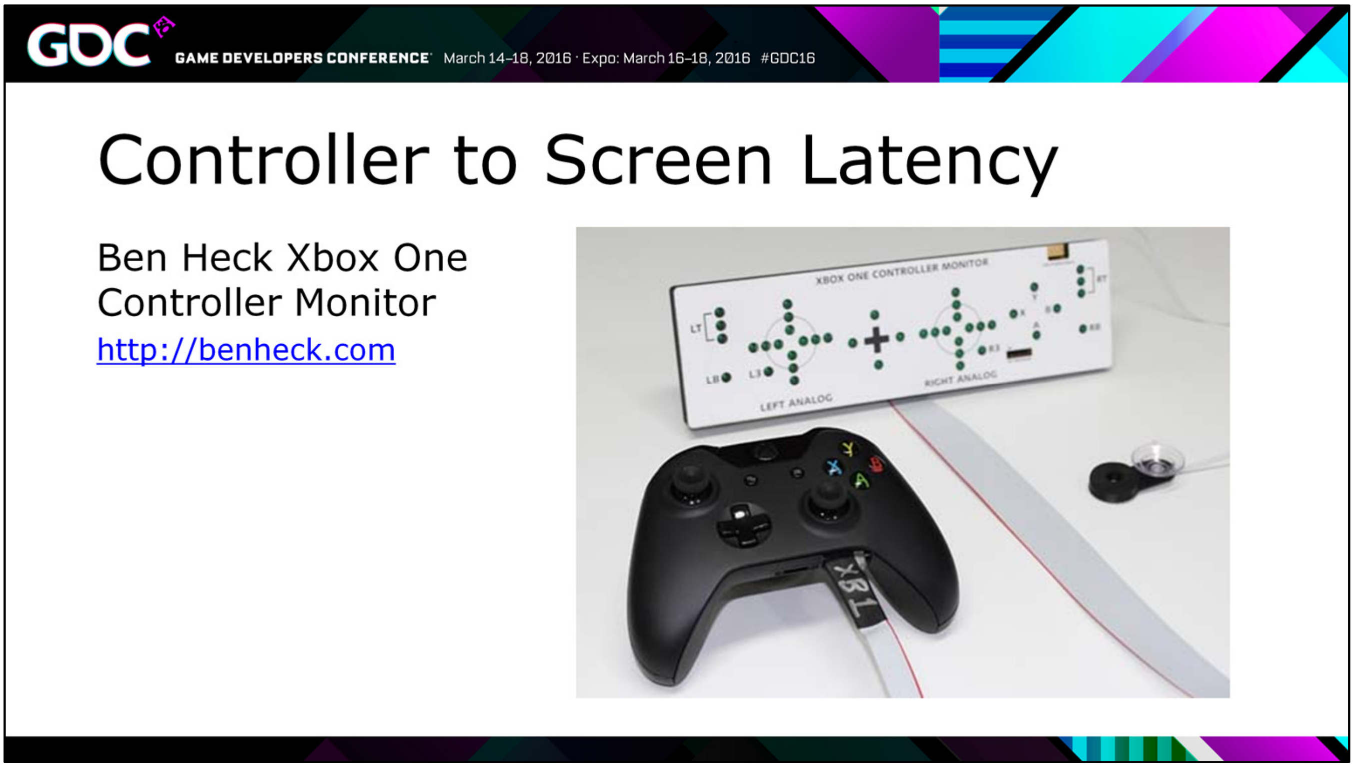
GDC GAME DEVELOPERS CONFERENCE March 14–18, 2016 · Expo: March 16–18, 2016 #GDC16

Blind A/B Testing

We had a procedural test and a camera test but, because this is Call of Duty, a change in hit-detection code is still a scary thing to deploy; Players are very sensitive to the feel of the game.

We did a blind A/B test with testers. It revealed that the new netcode was a big improvement from the player's perspective.

Now with all three – a classic procedural test, a camera test and an A/B test – Treyarch felt confident that they were releasing an improvement.



Controller to Screen Latency

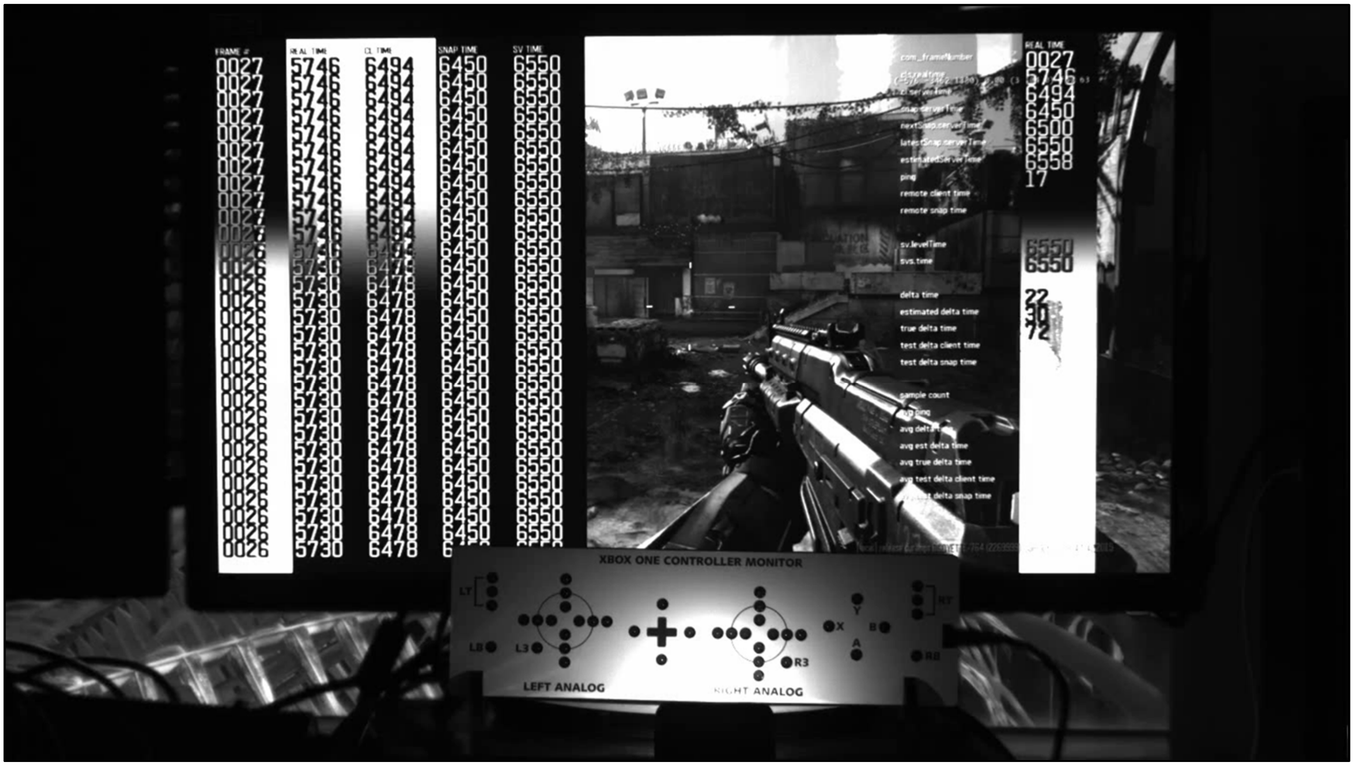
Ben Heck Xbox One Controller Monitor

http://benheck.com

Controller to Screen Latency is the delay between the button press and the reaction on screen, locally.
It will greatly contribute to the overall feeling of latency.

Measuring Controller to Screen Latency with a normal controller
Put the controller in the camera’s shot and tap the button. But it is hard to tell in the video footage when is the exact moment the finger pressed the button. Especially since the recording is often dark and at a low resolution. It still gives a ‘ballpark’ idea.

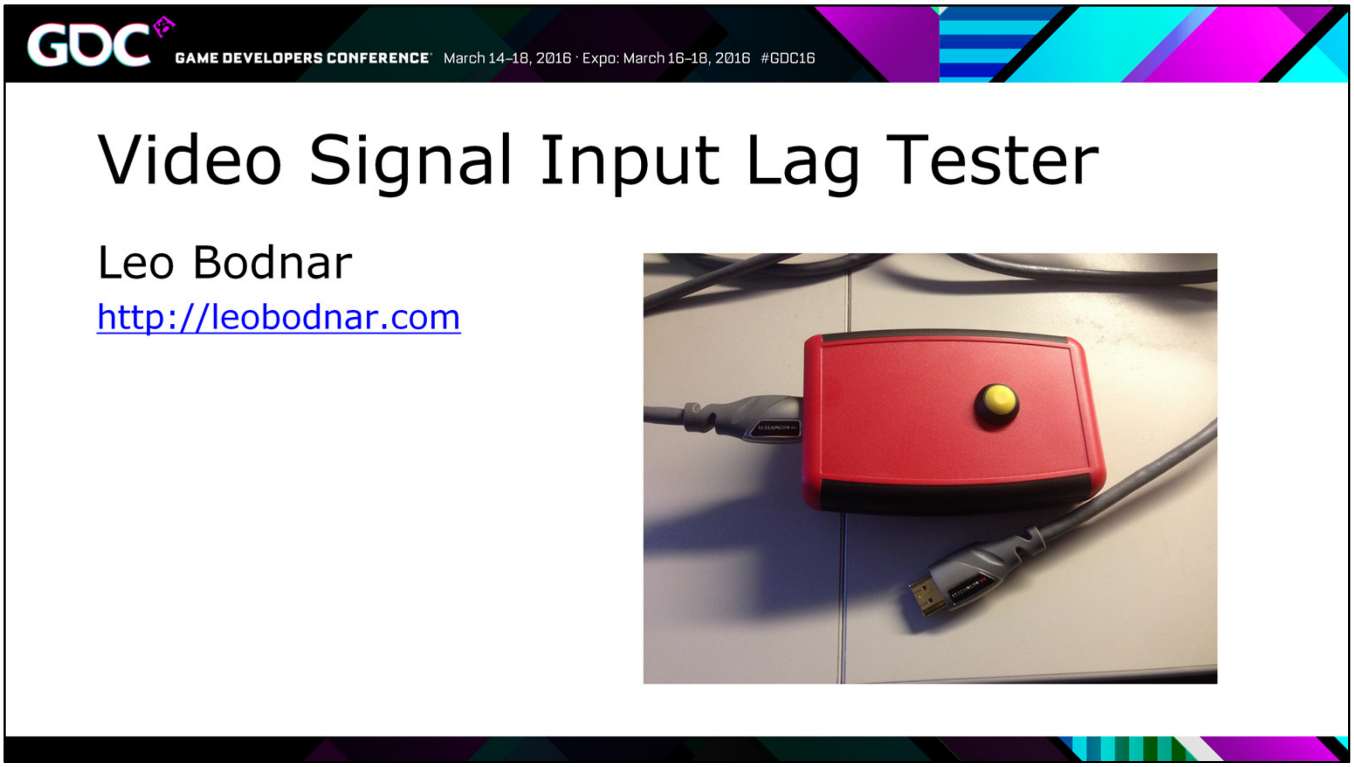
Measuring with a Ben Heck Xbox One Controller Monitor
Goal: To get a more precise measurement. The board has LEDs that lit up when pressing buttons on the Xbox controller. Recording both the board and the screen with the high-speed camera, you can then more precisely count the frames between the moment the LEDs lit up and the action on screen. The board connects to a modified Xbox controller to read the signal straight from the source.



Video – Measuring with a Ben Heck Controller Monitor

Testing various action:

- Jumping (face buttons)
- Moving (analog sticks)
- Shooting (analog triggers)



Video Signal Input Lag Tester

Leo Bodnar

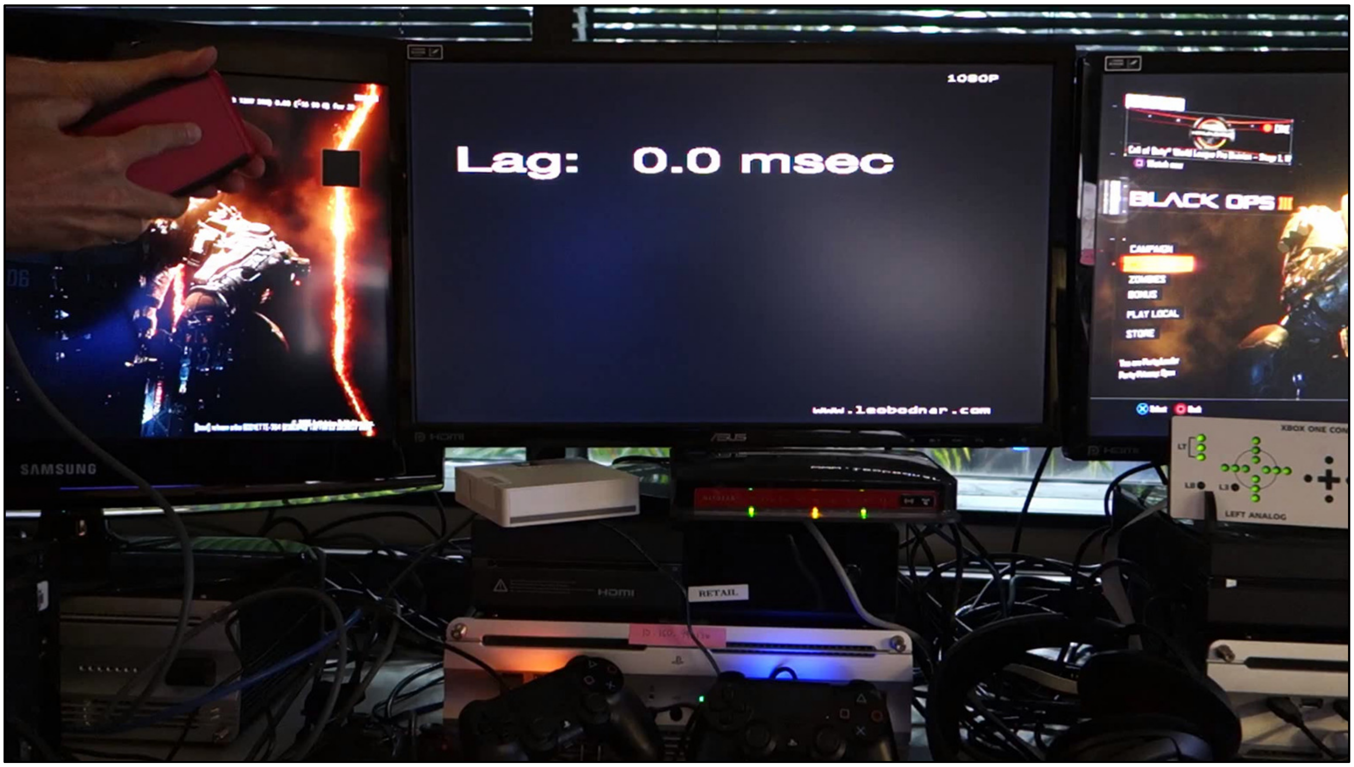
http://leobodnar.com

Measuring the Screen’s Latency with Leo Bodnar’s Input Lag Tester
Goal: To give you an idea of how much latency the screen is accounting for.

It’s a simple and popular device for measuring a screen’s latency.

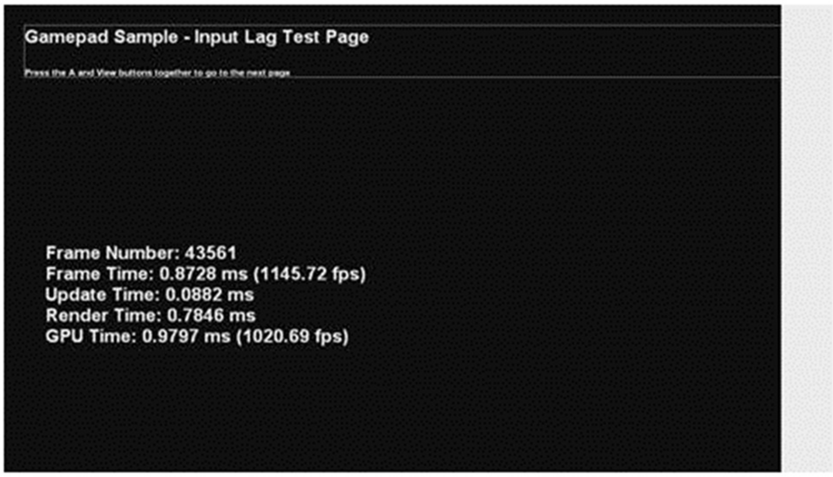
- Used by monitor reviewers.
- Just connect it to an HDMI input and put the device over the white bars on the screen.
- The measurement will be displayed on the screen.
- It works using a photosensor.

Measure at the middle of the screen since that’s where you take your measurement with the camera.



Video – Measuring with the Leo Bodnar Tester.

Platform’s Minimum Latency



Gamepad Sample - Input Lag Test Page

Press the A and View buttons together to go to the next page

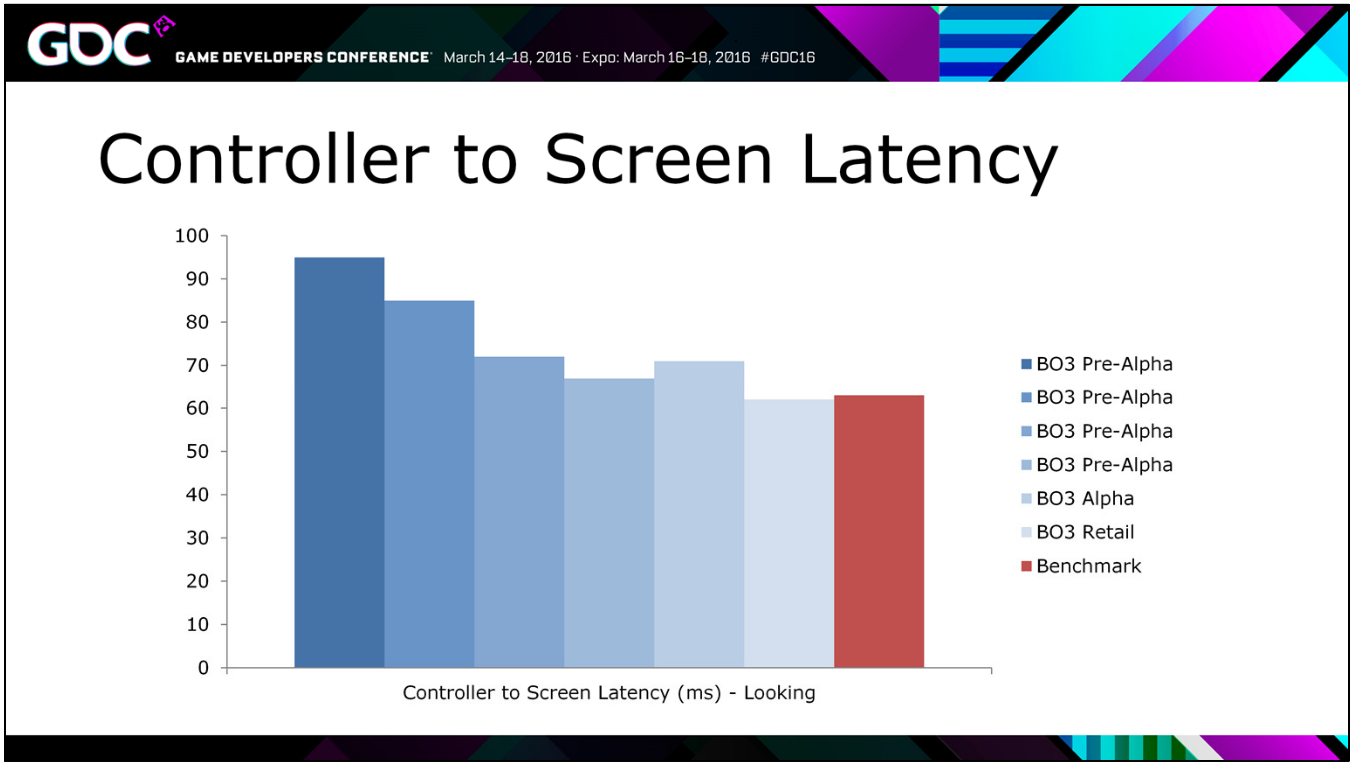
```
Frame Number: 43561
Frame Time: 0.8728 ms (1145.72 fps)
Update Time: 0.0882 ms
Render Time: 0.7846 ms
GPU Time: 0.9797 ms (1020.69 fps)
```

Testing the baseline latency of a platform

Goal: Know what is the absolute minimum Controller to Screen Latency you could get on a platform. And see how much your game is accounting for.

Approach: Write a very simple program that is just polling the inputs as fast as possible and refreshing the screen.

- Test with unlimited frame rate, no-vsync: The lowest possible.
- Test with different frame rates.
- Test with different V-sync modes.

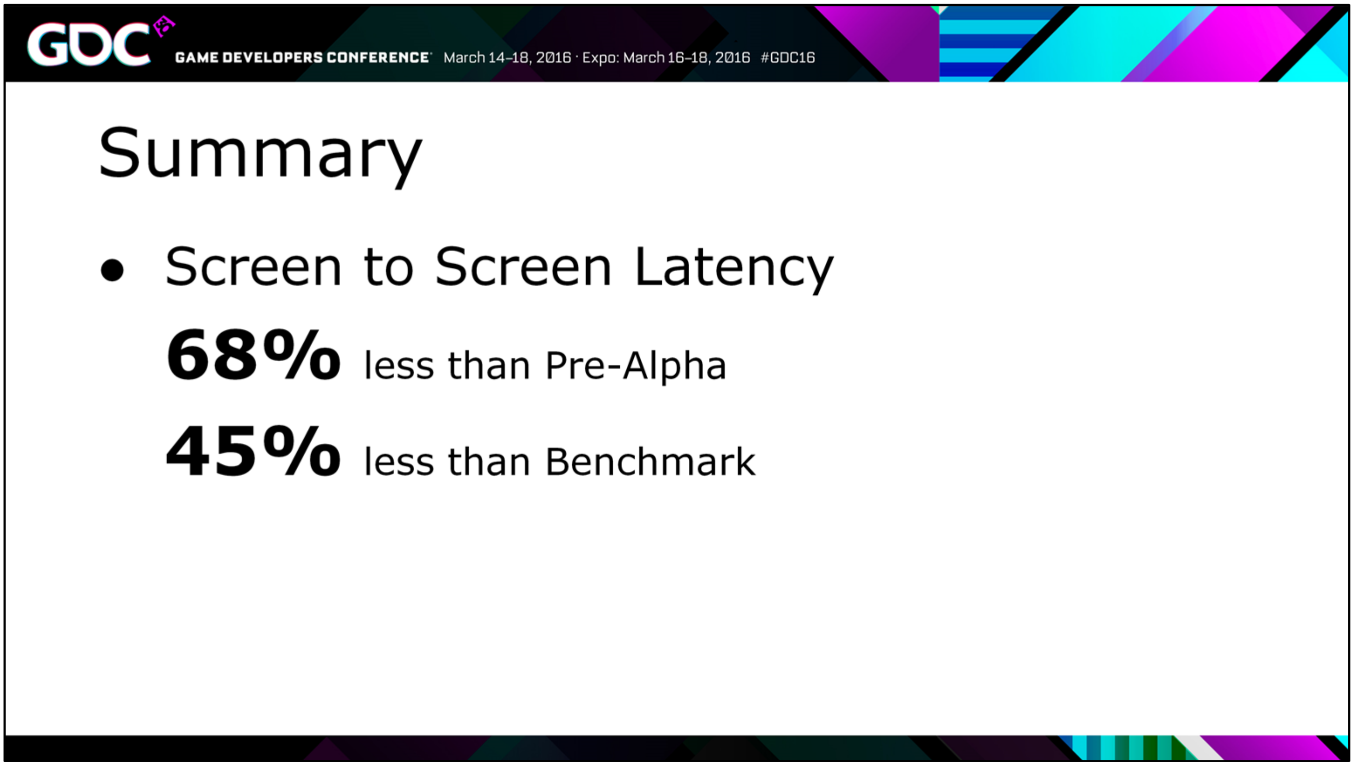


Controller to Screen Latency

Initial tests showed that we had +35ms of excess latency compared to the benchmark. This confirmed it was a real problem.

Not diving into what we changed in this talk. It could be a whole talk.

But it’s an important tool to have to be able to validate assumptions.

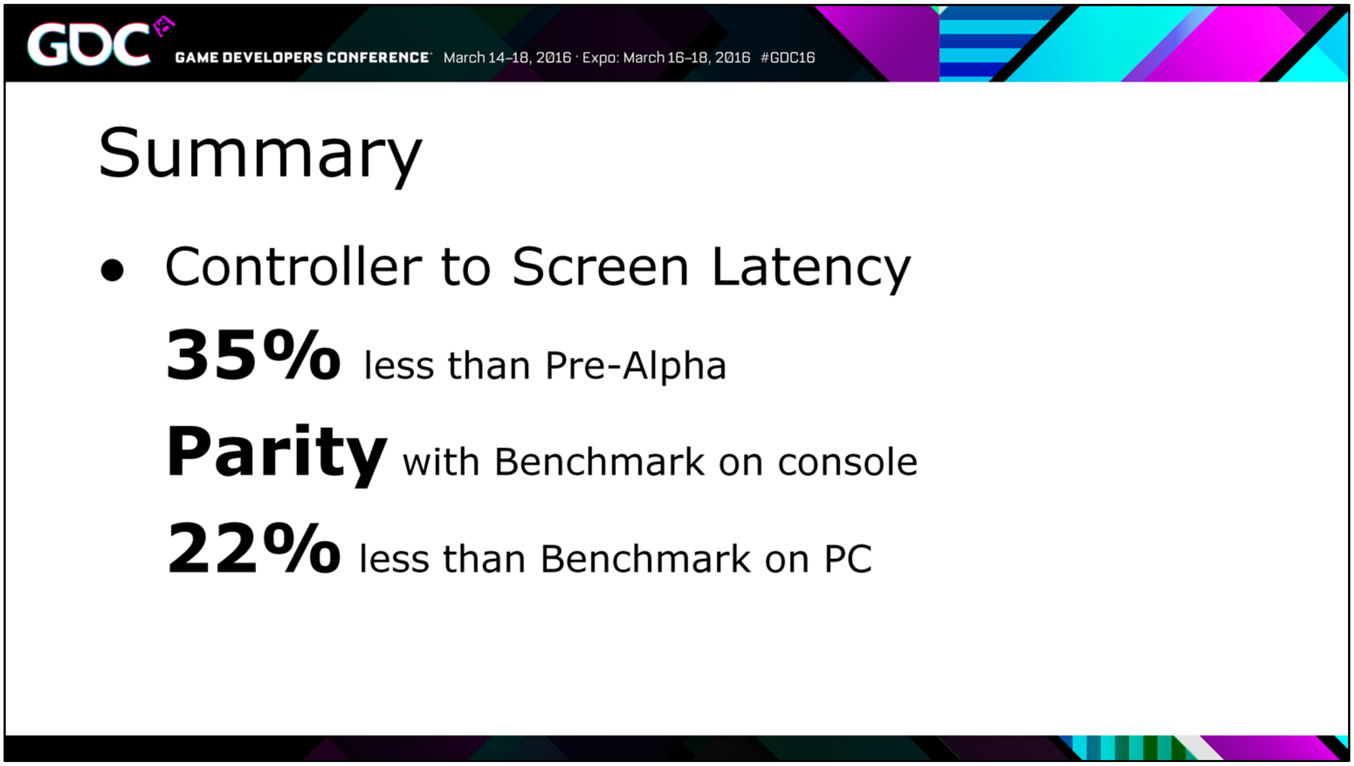


Summary

- Screen to Screen Latency

68% less than Pre-Alpha

45% less than Benchmark



Summary

- Controller to Screen Latency

35% less than Pre-Alpha

Parity with Benchmark on console

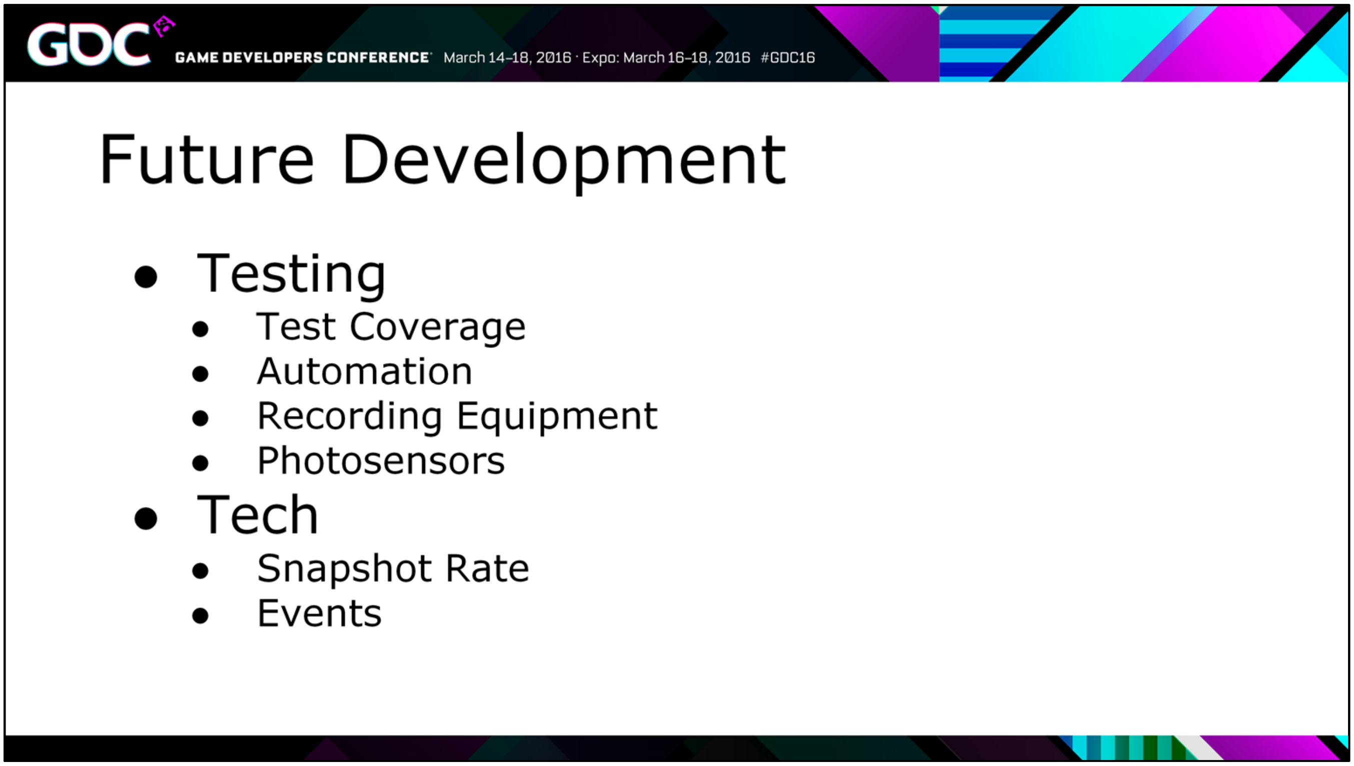
22% less than Benchmark on PC

Summary

- Improved hit-detection fidelity

  9/10

  “I had a great experience”



Future Development

- Testing
  - Test Coverage
  - Automation
  - Recording Equipment
  - Photosensors
- Tech
  - Snapshot Rate
  - Events

More test cases, audio latency, etc.

Automation with the build system. Catch regressions same day.
Use image analysis?

Recording equipment improvements for accuracy and workflow.
Experiment with dual channel video recorders.

Photosensors: Very promising and automatable.

Tech:
Balancing the cost of snapshot rate with worthwhile improvements.
Don’t just try to get pretty numbers (60 megapixels!!) – Measure the actual perceived improvement.
Use smaller faster events (not entire snapshots) for some effects. Watch out for anything impacting the hit detection fidelity.

FIGHTING LATENCY

ON

CALL OF DUTY

BLACK OPS III

BEN GOYETTE

ENGINEER AT ACTIVISION

I showed you how we fought latency using an high speed camera. If there’s one takeaway, I hope is that it made you curious to check your game.
You don’t need a fancy mocap camera to get an idea though. With just an iPhone, you could see if your game really is as fast as your tools are telling you.