Frostbite

RENDERING ENGINE ARCHITECTURE CONFERENCE

REAC24

Testing Rendering Code at Frostbite

Jon Valdés

Frostbite

Note: These slides don’t match the actual presentation exactly.

I did a lot of video editing after the fact, so take these slides as a rough approximation

Who am I?

Jon Valdés

Last 8 years at Frostbite Rendering in the Image Quality cell

Worked on FrameGraph, GI, post-processing, strand hair, a bunch of internal tooling...

Lately also engineering manager

About this talk

This isn’t a talk on The Right Way to Test™

It’s a talk on what we’re currently doing and what we’ve learned so far

If you think your system is better than ours, we’d like to learn from you.

***Please present it*!**

Required reading

Aras Pranckevičius

- Testing graphics code https://aras-p.info/blog/2007/07/31/testing-graphics-code/
- Testing Graphics Code, 4 years later https://aras-p.info/blog/2011/06/17/testing-graphics-code-4-years-later/

Required reading

Bart Wronski

- How (not) to test graphics algorithms
  https://bartwronski.com/2019/08/14/how-not-to-test-graphics-algorithms/

Required reading

Keith Stockdale from Rare gave this **fantastic** talk about their shader testing system earlier this year

https://schedule.gdconf.com/session/automated-testing-of-shader-code/899160

What is Frostbite

Why we care about testing

What is Frostbite

Electronic Arts' internal game engine

Every game team within EA can use Frostbite if they want

We provide the technology, and help game teams use it

Frostbite’s mission

Help games ship.

The kind of games we make

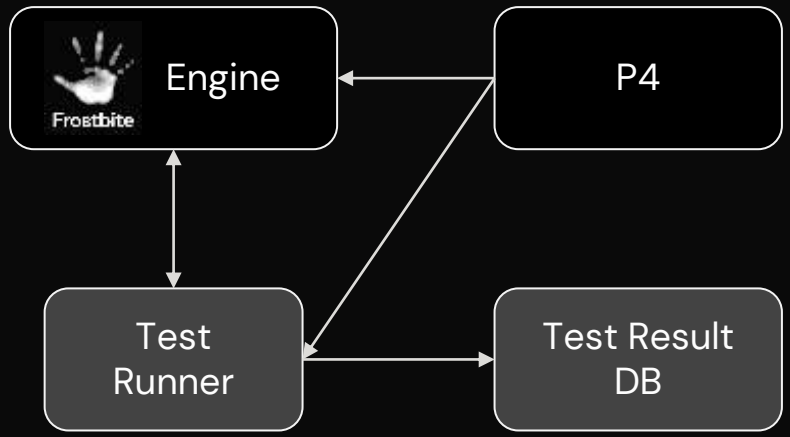
Stability is one of our top concerns

A stable engine is a top priority for our game teams.
(Particularly for the ones that are shipping a game every year!)

- “This used to work on our last game, now it doesn’t”
- “This was faster last year”
- “We just tested features A, B, C and D together, and that breaks feature E”
- “This new feature doesn’t work as expected”

Test infrastructure at Frostbite

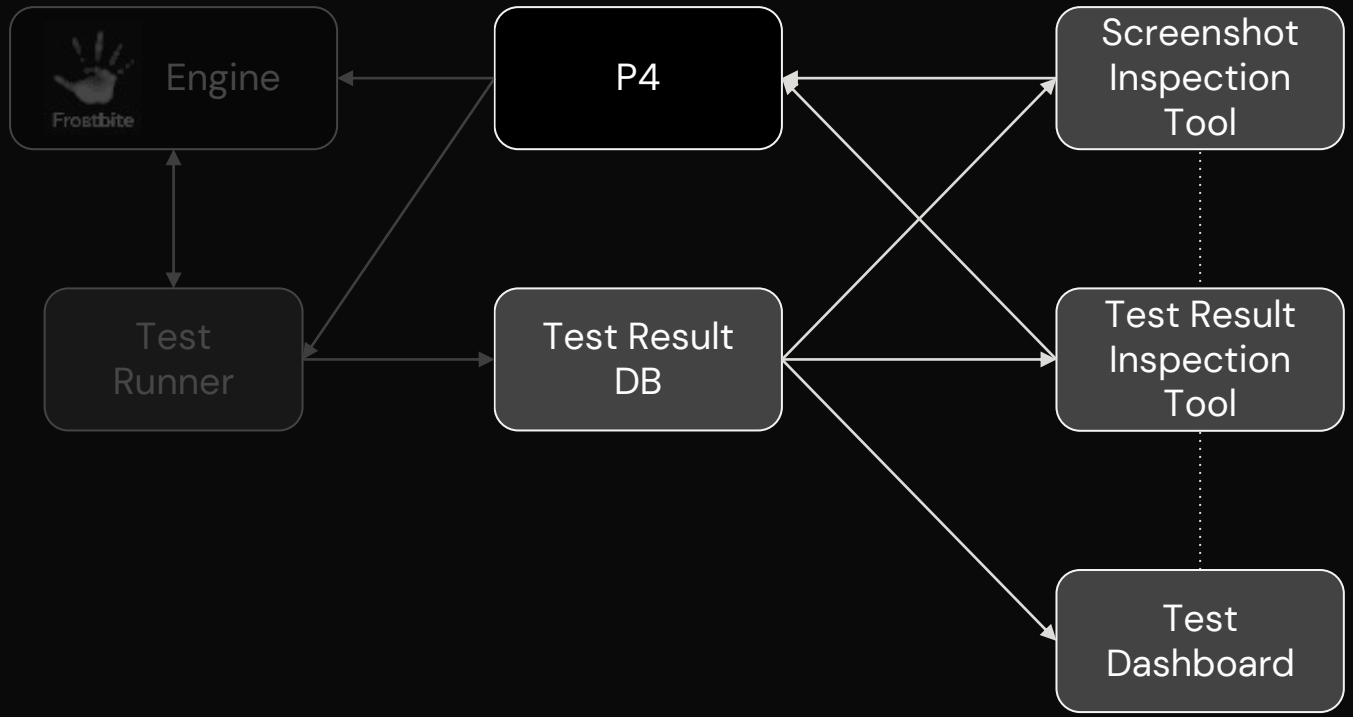
Current infrastructure



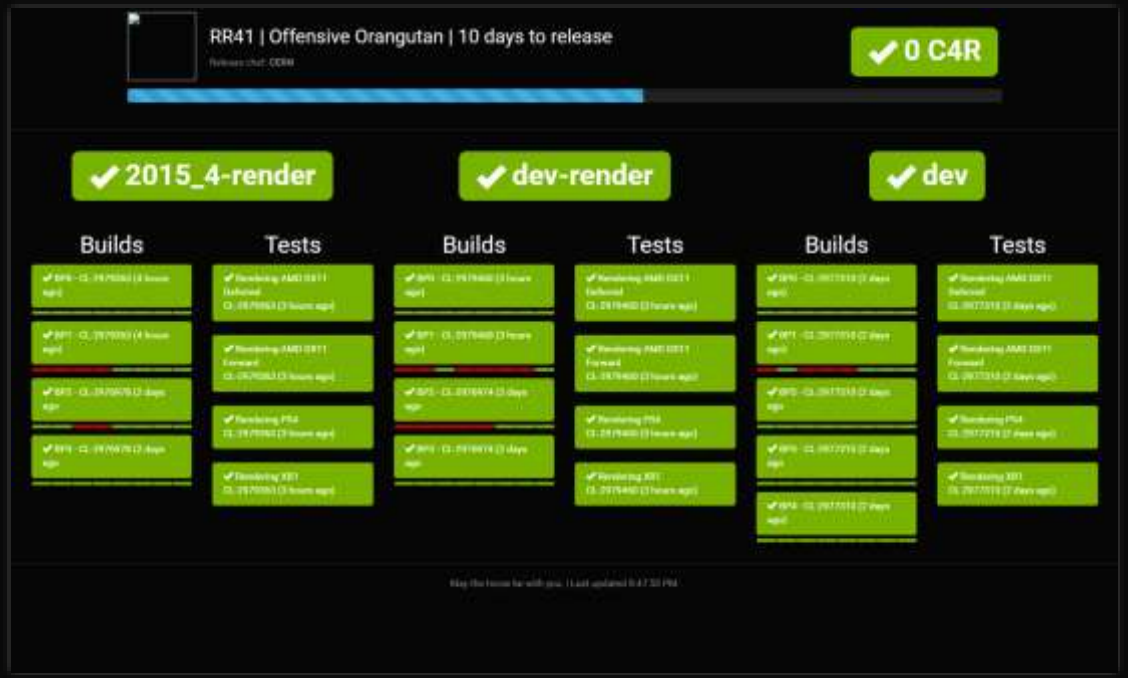
Code, test data, screenshots **always** live in same P4 workspace

Strong Code ⇔ Data interdependency

Current infrastructure



Testing Dashboard in 2016



- 3 branches
- 4 test suites: XB1, PS4, PC DX11 Forward + PC DX11 Deferred

Testing Dashboard in 2024

PC DX12, PC Vulkan, XBSX, XB1, XB1X, PS4, PS4Pro, PS5, Switch, iOS, Android

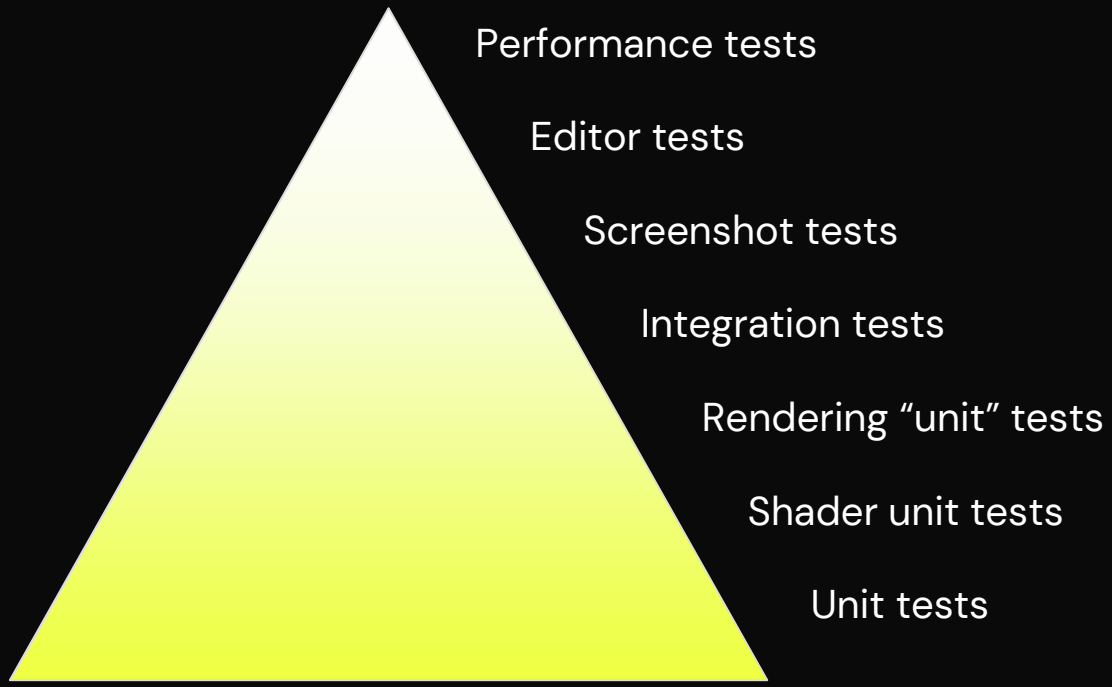
>1000 test suites

>200 SEs at Frostbite

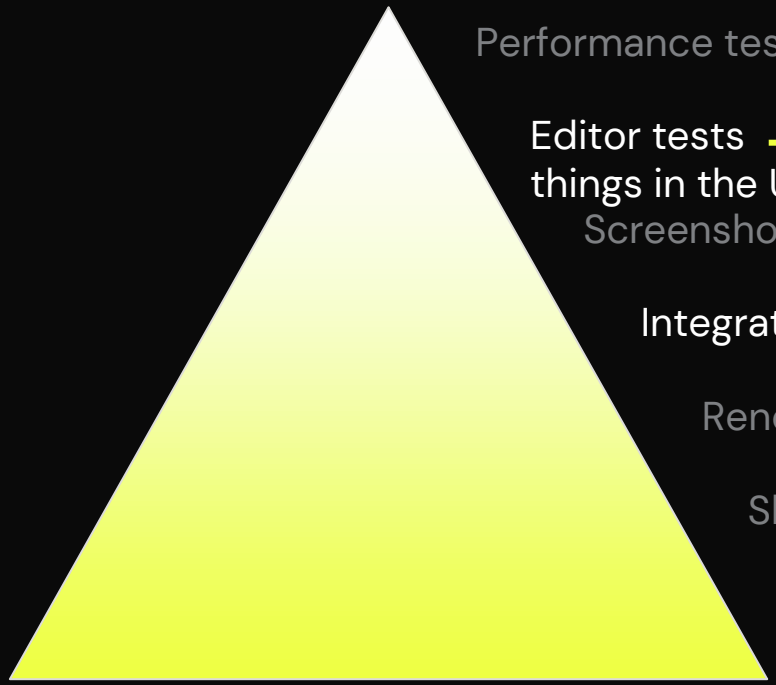
Pre- & post-submission tests

Test types

Automated tests at Frostbite



Automated tests at Frostbite



Performance tests

Editor tests → "Click on things in the UI"

Screenshot tests

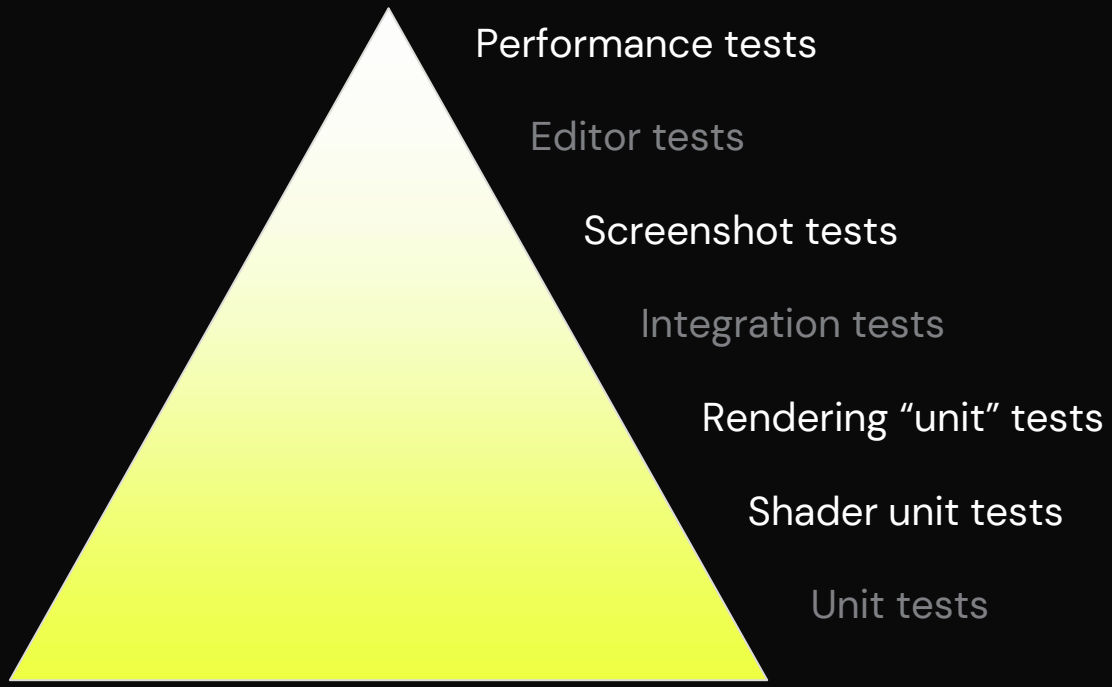
Integration tests → "Load level and do stuff"

Rendering "unit" tests

Shader unit tests

Unit tests → GoogleTest

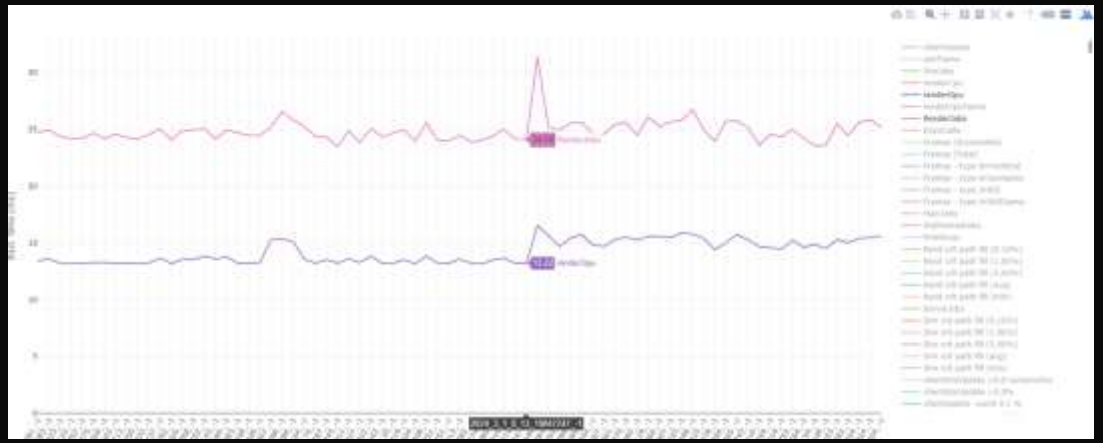
Automated tests at Frostbite



Performance Tests

They run regularly in the farm

- We get alerts whenever performance deviates from history (both upwards and downwards)
- Both CPU and GPU markers
- Significant manual analysis/triaging, but it's useful to avoid perf drift



Shader Unit Tests

```
[Test]
public void Test_frustumCullingCone()
{
    var result = CreateComputeHarness().ExecuteShaderFunction<bool>(
        "Systems/Lighting/Explicit/frustumCullingCone.hlsl",
        "frustumCullingConeWrapper",
        new Vec3<float>(0, 0, 0),
        1.0f,
        new Vec3<float>(0, 0, 1),
        0.5f,
        new Vec4<float>(0, 0, 1, 0),
        new Vec4<float>(0, 0, 1, 0),
        new Vec4<float>(0, 0, 1, 0),
        new Vec4<float>(0, 0, 1, 0),
        new Vec4<float>(0, 0, 1, 0),
        new Vec4<float>(0, 0, 1, 0),
        new Vec4<float>(0, 0, 1, 0),
        new Vec4<float>(0, 0, 1, 0));

    Assert.That(result, Is.EqualTo(false));
}
```

Just like C# unit tests, but invokes individual HLSL functions.

Uses WARP to execute shader code in the CPU, which removes GPU differences (And we can run it on any farm VM)

Not using them a lot yet, but they’re pretty neat :)

Rendering “Unit” Tests

```
void testCullRadixSort64UnsortedLarge(RenderTestContext* context)
{
    RadixSortInput unsortedData = {
        514, 423, 313, 898, 7238, 1, 202, 665,  51, 4233,  33, 8918,  72,   2, 1202, 6635,
        532, 433, 332, 828, 7824, 5, 222, 661,  17, 4222,  31, 8383,  22,   6, 1112, 6660
    };

    run(context, unsortedData);
}

void run(RenderTestContext* context, const RadixSortInput& unsortedData)
{
    FrameGraph frameGraph(m_frameGraphArena, "TiledLightingRenderTest");
    addLightCullRadixSortTestPass(frameGraph, unsortedData, m_outputResource);
    frameGraph.execute(context->getCommandRecorder());

    context->getReporter().readBackBuffer<u32>(m_outputBuffer, [=](const u32* radixSorted) {
        for (uint arrayIt = 0; arrayIt < unsortedData.dataCount-1; ++arrayIt)
        {
            FB_ASSERT_DESC(radixSorted[arrayIt+1] >= radixSorted[arrayIt],
                "RadixSort of " << unsortedData.dataCount << " entries has failed: entry " << nextEntry <<
                " (" << radixSorted[arrayIt+1] << ") is less than entry " << arrayIt
                << "(" << radixSorted[arrayIt] << ").");
        }
    });
}
```

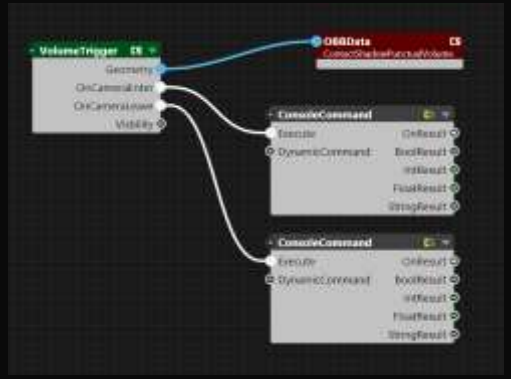
No level, no scripting.
No infrastructure

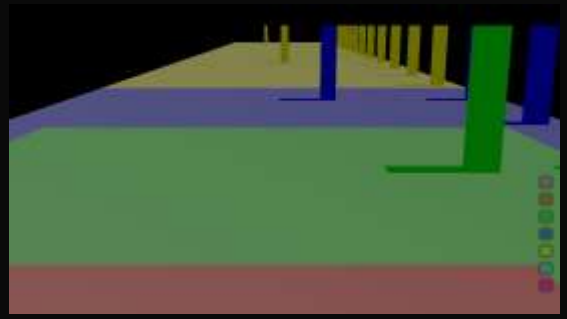
Can `memcmp` images/buffers, or manual asserts

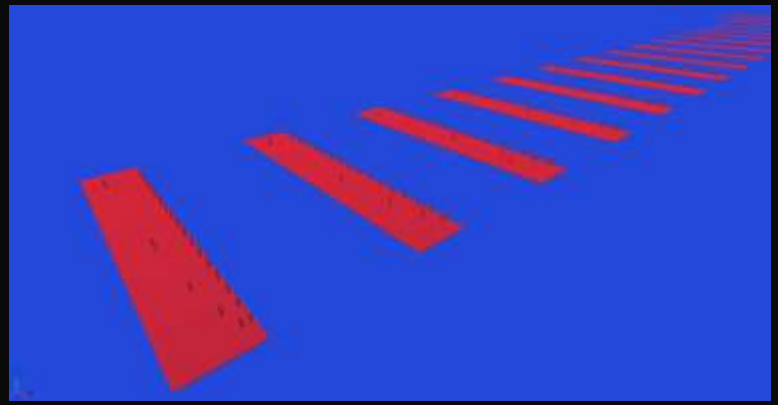
Screenshot tests

Multiple cameras per level

- Switch to new camera
- Execute setup commands
- Take screenshot
- Send screenshot to test harness
- Execute teardown commands
- Repeat







Comparing images with thresholds

Industry standard - nVidia FLIP: https://github.com/NVlabs/flip

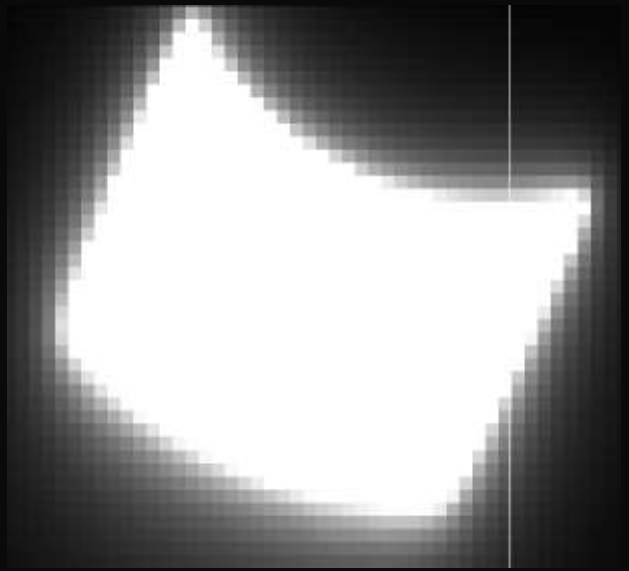
We don't use it. We use a ***bad*** per-pixel metric

```
float normalizedDiff = (diff.r + diff.g + diff.b) / 7.65; // No idea where this comes from.
```

Then the score for the whole image is the max **`normalizedDiff`** of all pixels

It's... not great. But in practice it's... uuuuh... *fine*?

Comparing images with thresholds

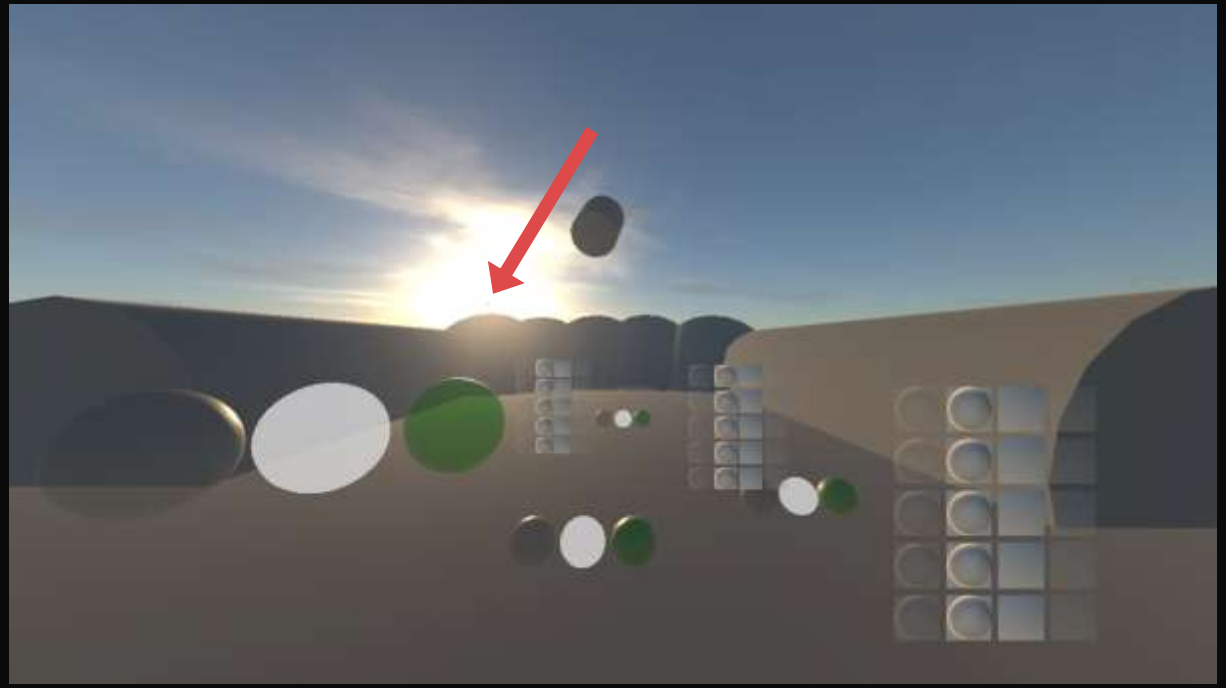


We get away with it because of ***extremely*** stringent thresholds.

We try to lower the thresholds as much as possible

And the test is per-pixel

Comparing images

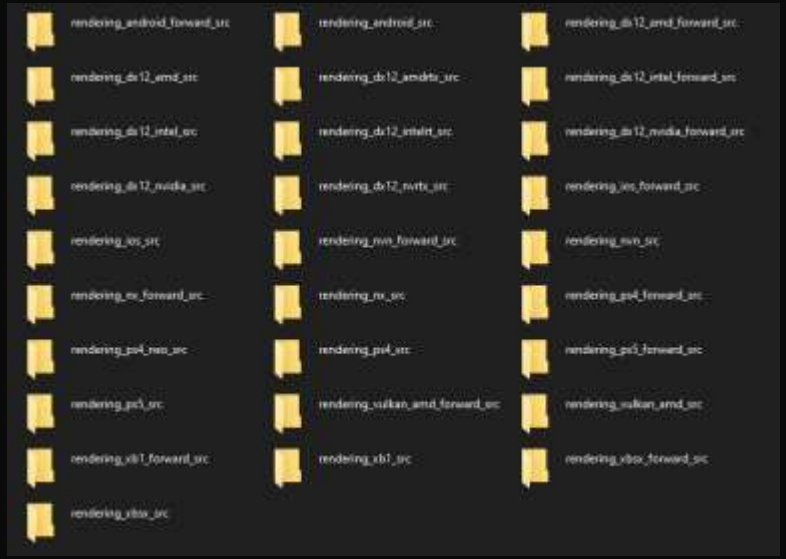


“But a single different pixel will make the test fail!!”

Yes!

A single bad pixel can be terrible

Comparing images



"But what about HW differences??"

Different reference images per platform/IHV!

We currently have ~~32849~~ 33913 *reference screenshots*

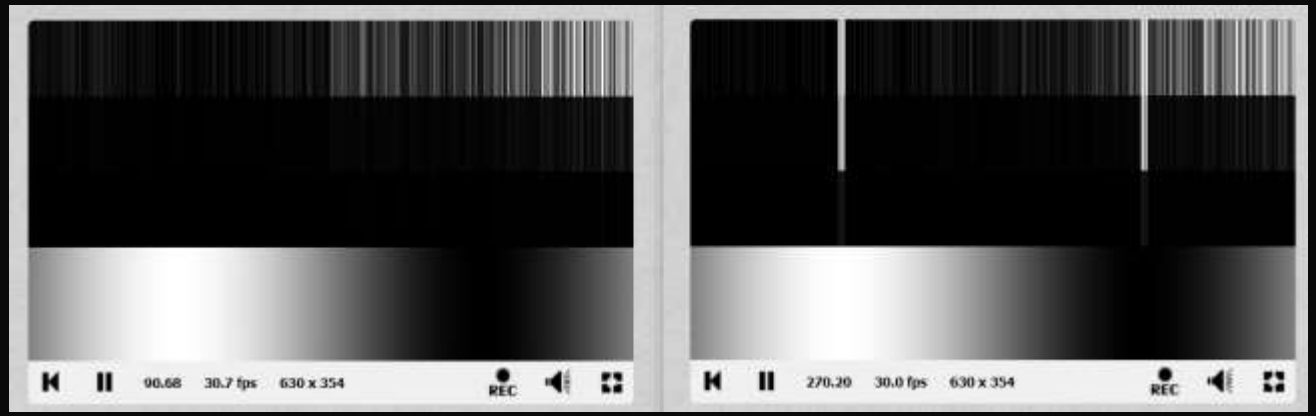
Pain threshold warning

“Just make your test HW-independent!”

Many things are IHV/driver/compiler/OS-dependent.

- Biggest one: Texture filtering
- Fun one: Transcendental function precision
  (DX only specifies min precision)

https://www.shadertoy.com/view/dtlyD8



NVIDIA 2080TI

Intel Iris Xe

RenderTests

```
void testCullRadixSort64UnsortedLarge(RenderTestContext* context)
{
    RadixSortInput unsortedData = {
        514, 423, 313, 898, 7238, 1, 202, 665,  51, 4233,  33, 8918,  72,   2, 1202, 6635,
        532, 433, 332, 828, 7824, 5, 222, 661,  17, 4222,  31, 8383,  22,   6, 1112, 6660
    };

    run(context, unsortedData);
}

void run(RenderTestContext* context, const RadixSortInput& unsortedData)
{
    FrameGraph frameGraph(m_frameGraphArena, "TiledLightingRenderTest");
    addLightCullRadixSortTestPass(frameGraph, unsortedData, m_outputResource);
    frameGraph.execute(context->getCommandRecorder());

    context->getReporter().readBackBuffer<u32>(m_outputBuffer, [=](const u32* radixSorted) {
        for (uint arrayIt = 0; arrayIt < unsortedData.dataCount-1; ++arrayIt)
        {
            FB_ASSERT_DESC(radixSorted[arrayIt+1] >= radixSorted[arrayIt],
                "RadixSort of " << unsortedData.dataCount << " entries has failed: entry " << nextEntry <<
                " (" << radixSorted[arrayIt+1] << ") is less than entry " << arrayIt
                << "(" << radixSorted[arrayIt] << ").");
        }
    });
}
```

- Exact comparisons
- Extremely robust
- Short-term pain
- Long-term bliss

Screenshot Tests

- Inexact comparisons (thresholds)
- Range: [brittle, robust]
- Short-term bliss
- Long-term pain

The screenshot flame wars

Screenshot Wars

```
ASSERT_EQ(blend(white, black, 0.5), grey);
```

Screenshot Wars

“Is there banding on smooth gradients in our volumetrics?”

“Is TAA converging well after a disocclusion?”

“Is our sky the right shade of blue?”

“Is the noise pattern on GTAO visible?”

Screenshot Wars

1. Setting up complex cases in code is sometimes not viable

1. **“Correct” is a moving target**. Often can’t mathematically validate correctness. Lots of rendering techniques are approximations. “Correct” changes as approximations get better (or worse!).

The ultimate reason

And, frankly,

Games have to ship

Ideal: Consider testing cost over system lifetime

Reality: This system needs to be in the game by next month

Sometimes it's better to have imperfect tests today than perfect tests two months from now

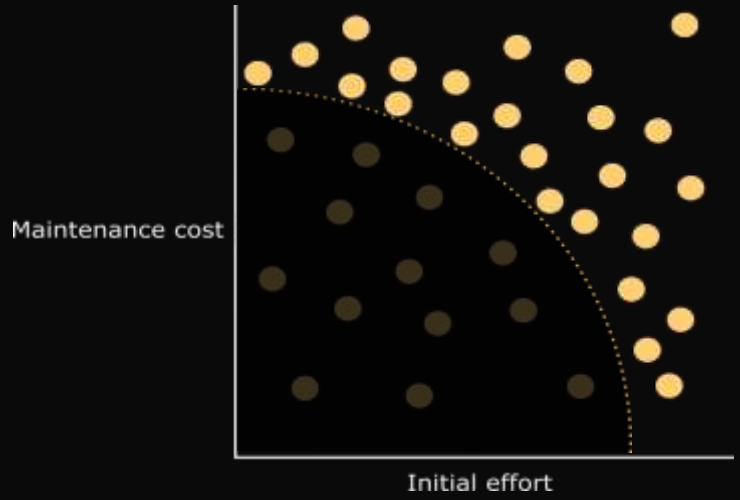
High-quality screenshot tests

Rendering Test Quality

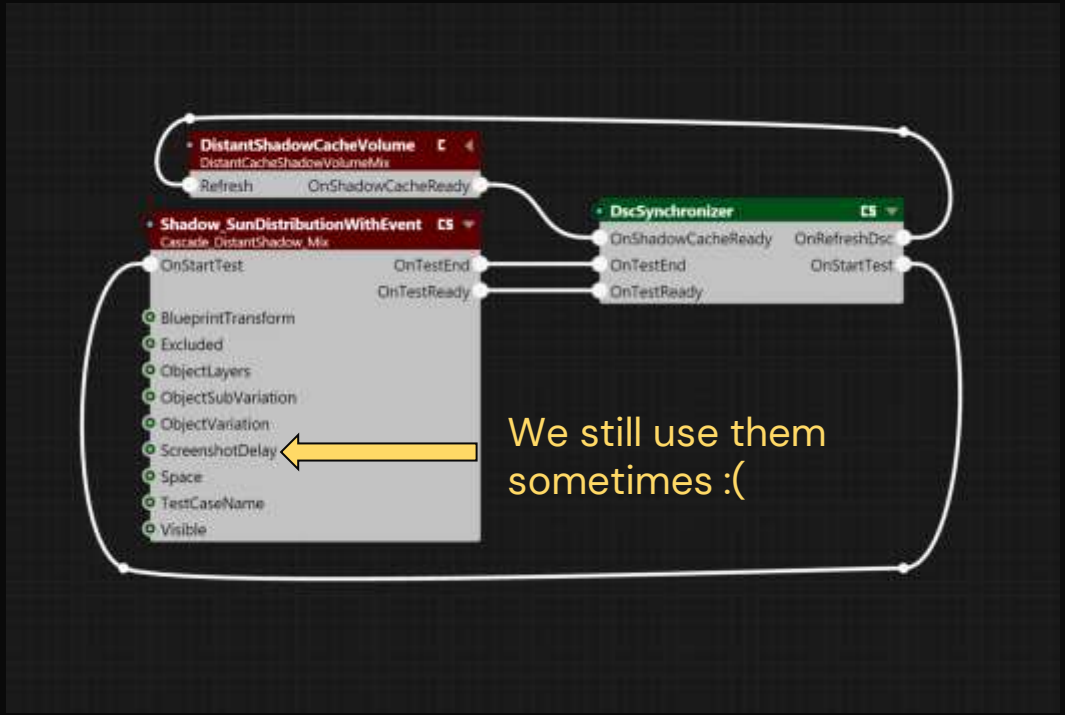
Haven't found a way to have low-effort, high-quality tests

Less long-term maintenance requires more initial effort:

- No reused data between tests
- Fully deterministic codepaths ("deterministic mode")
- Exact comparisons
- Decoupled test (only test 1-2 systems)



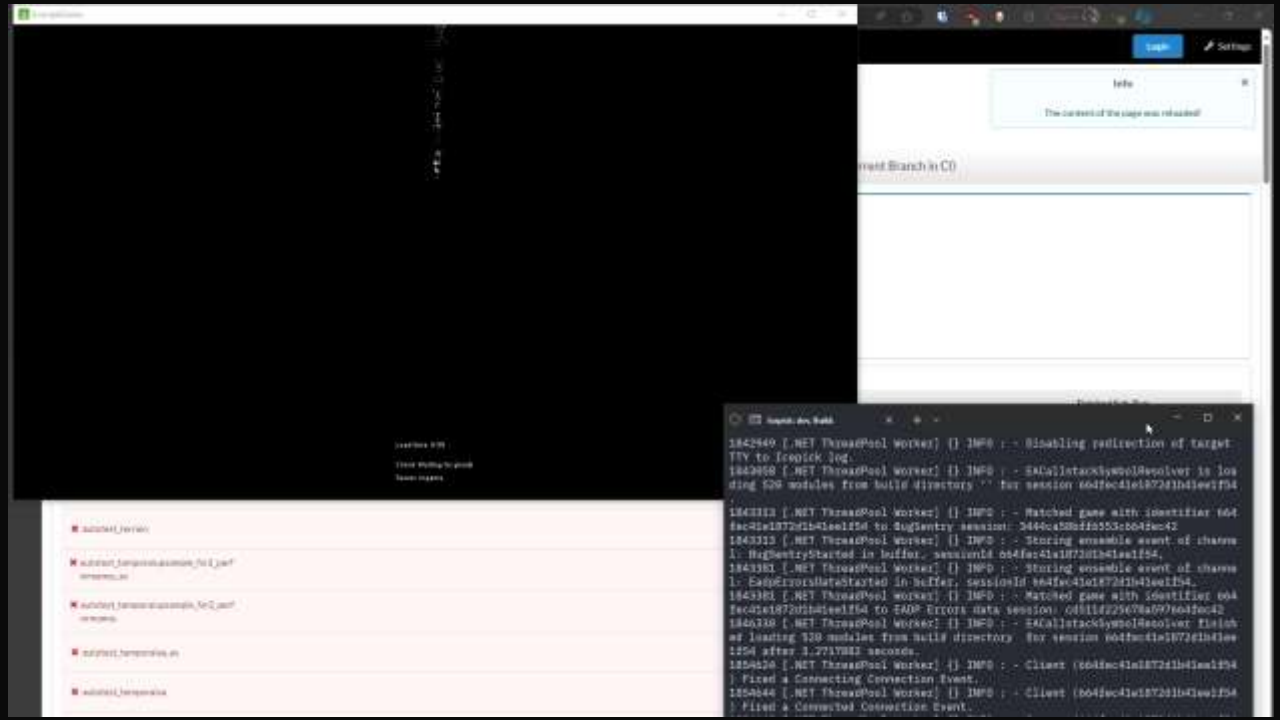
High-quality screenshot tests



Avoid arbitrary delays in tests.
Use events instead

“Oops, works locally but fails on the farm, because farm HW is oversubscribed and it runs slower”

Speed



Reducing delays allows faster iteration + CI

How to get to this point

We got here, and you can too

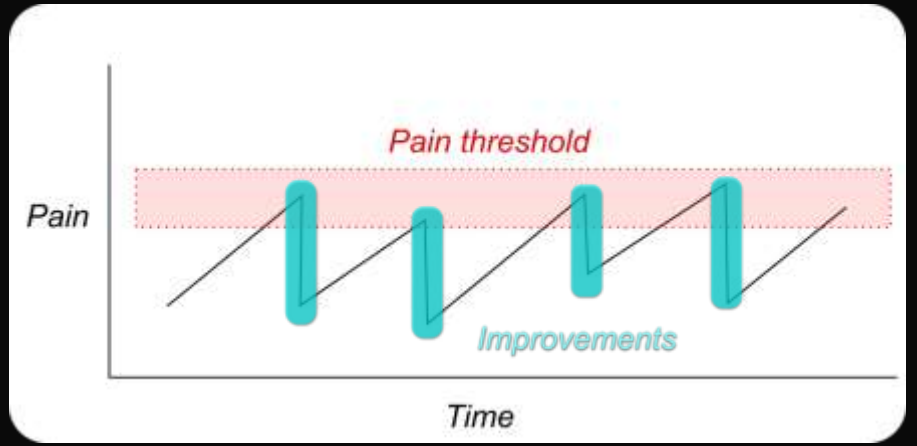
Pain-based development

Screenshot testing is easy short-term
But cost accumulates

Test, platform count always go up

Pain drives improvements

Overall pain remains ~constant

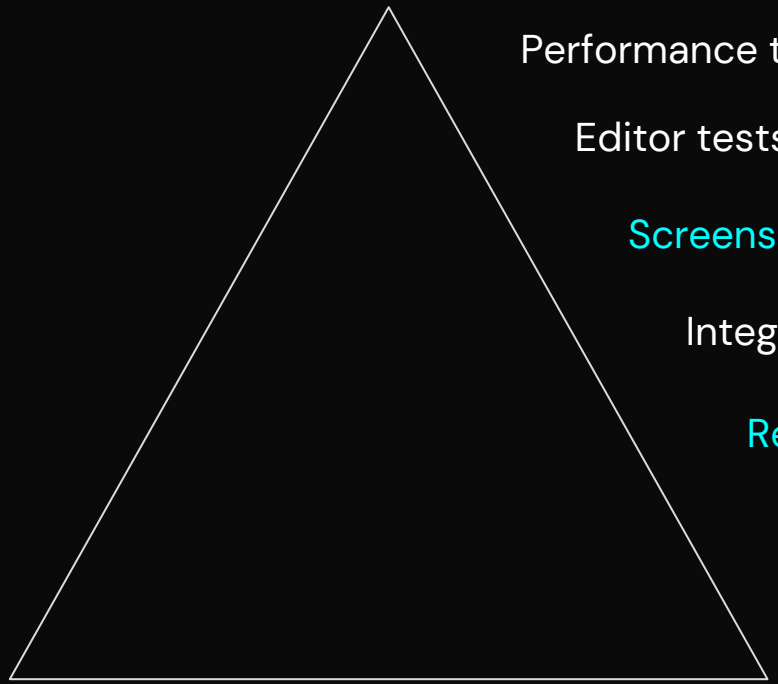


How to improve infrastructure

Top-down: convince leadership attacks problem

Bottom-up: single engineer

These started as grassroots projects



Performance tests

Editor tests

Screenshot tests

Integration tests

Rendering “unit” tests

Shader unit tests

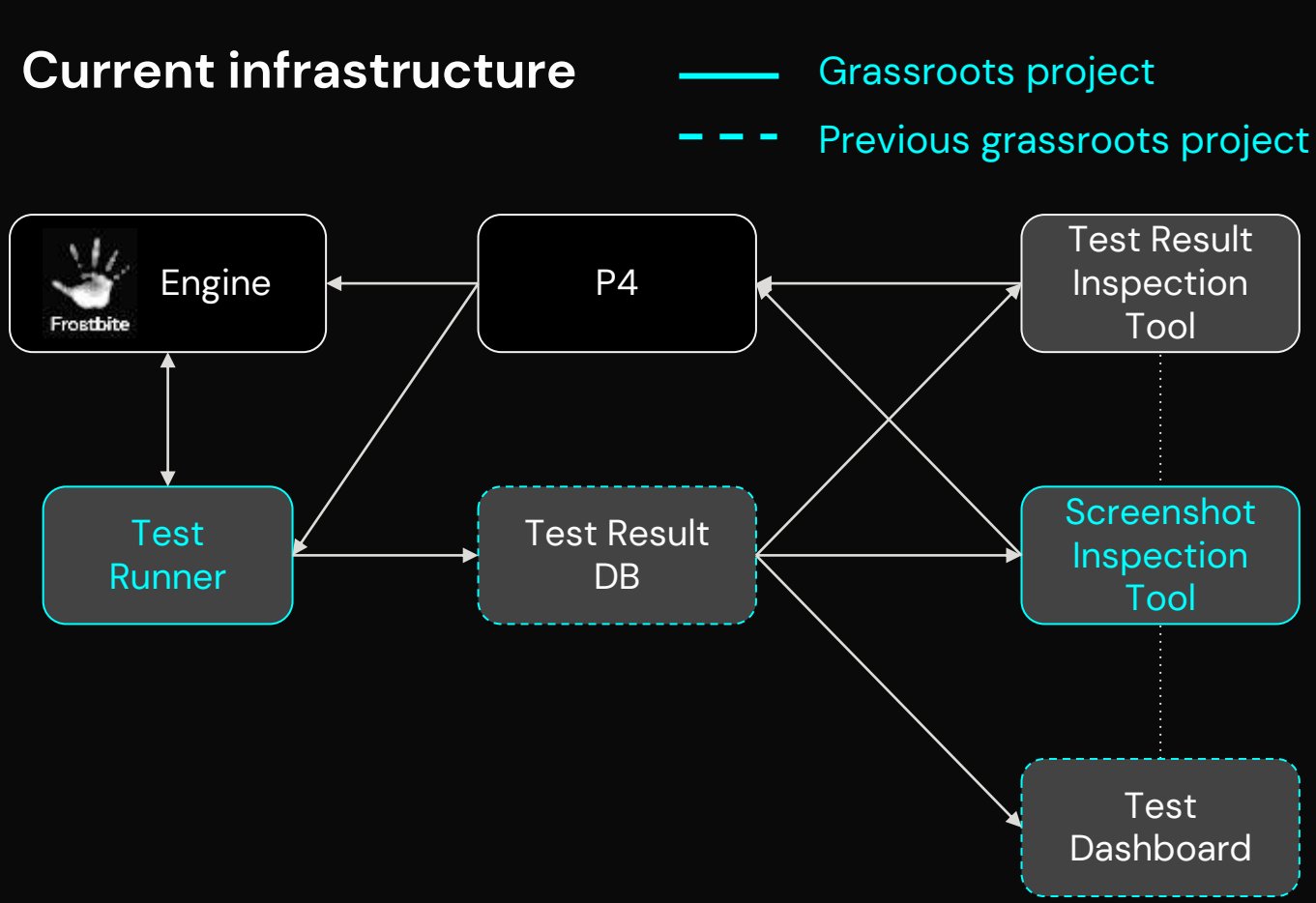
Unit tests

Pain-based development

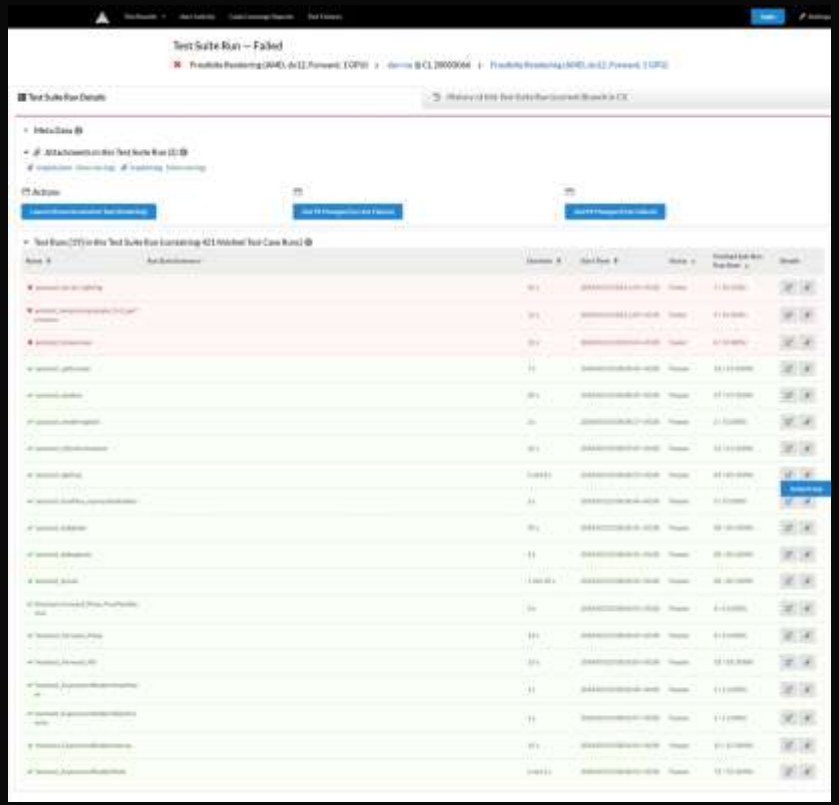
Pains

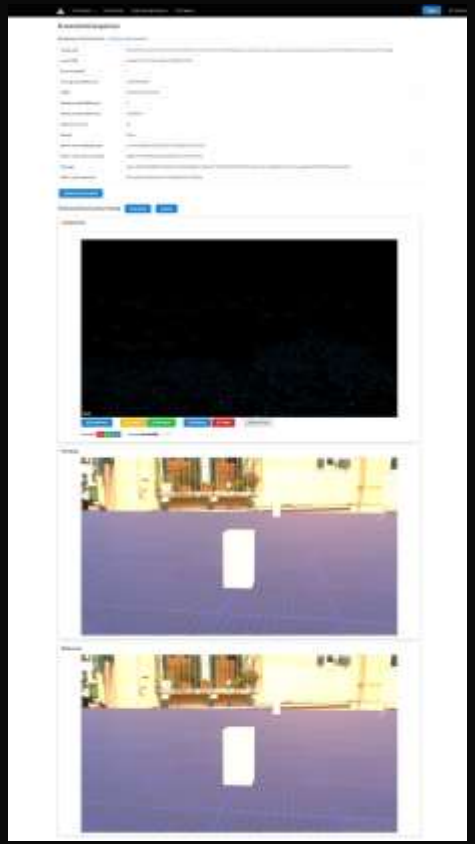
- Screenshot comparison is too lenient
- Inspecting/updating screenshots takes forever
- Generating new reference screenshots takes forever
- Too many false positives/negatives
- Screenshot comparison is too stringent

Current infrastructure



Test Result Inspection Tool



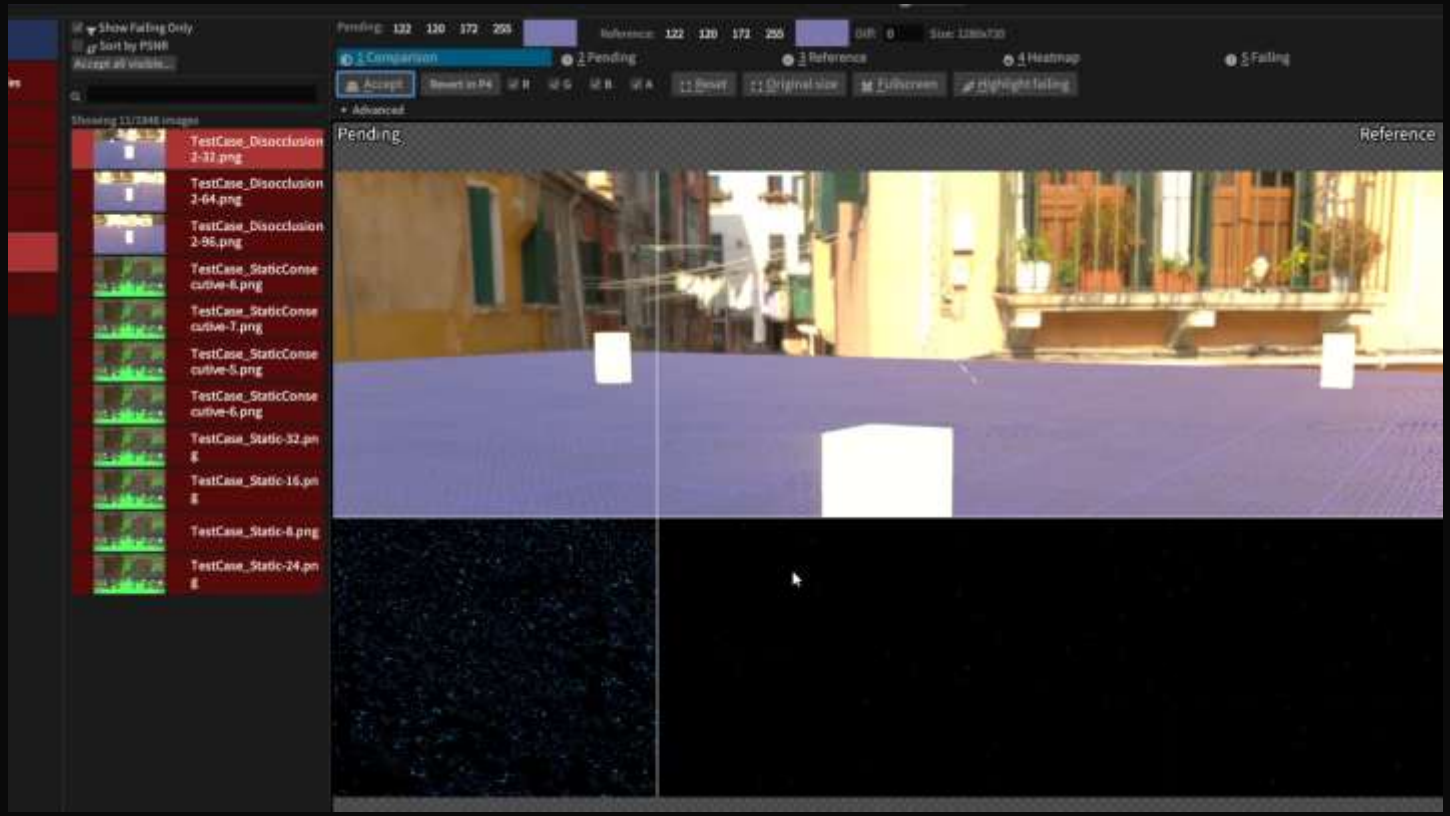


Test Result Inspection (Underling)

Tool built around a single goal: Make screenshot workflows fast

- Inspect screenshots quickly
- Update screenshots in bulk
- Ease screenshot comparison
- Once it loads, everything is in-memory, 60fps
- Power users can rely on keyboard shortcuts for almost everything

Underling usage



Is it worth it?

Is it worth it?

Yes! Confidence aids everything

- Tons of regressions caught before they impact anyone
- They help in debugging. Ex: White furnace test on strand hair helped us diagnose wrong sky values

Next steps

Because we’re far from done

Next steps: Comparing images

Some teams don’t care about single pixels

“We just want to know the object is more or less *there*”

Experimenting with Resolution Scaling right now
“Render at high res, then downscale”

Addendum

Infrastructure

- “Managed farm”: GPU HW, OS version, driver version, console SDKs tightly controlled
- Coordinated upgrades across branches

Next steps: running engine tests in game branches

- Ongoing work!
- Test data within the engine code

Future work: Comparing branches/platforms

- No good workflow to compare screenshots of different platforms/branches

- Workflow to accept “similar enough” screenshots on all platforms (but not all branches!!)

Conclusion

Testing for us always slightly to very painful

But you can make a difference

Δquality

Δengineer workflows

Δteam velocity

Thanks!

Thanks to everyone that has cared about our test infrastructure over the years!

Mick Beaver
Emma Fagerholm
Simon Taylor
Kyle Hayward
Jonathan Kidane
Homan Lam
Juan Parra
Alex Fry
Leszek Godlewski
Dan Elksnitis
Dave Cope
Johan Berg

Timo Qvist
Matt Fogarty
Sebastian Tafuri
Alex Matache
Bogdan Ionut Rotariu
Bryan Sefcik
Ben May
Bart Meijer
Gede Suparsa
Lydia Fang
Emily Benitez
Henrik “honk” Karlsson
Johnny Pak

Vani Choubey
Tayler Hopko
Hao Wang
Roman Gusev
Peter Lu
Joe Tomkins
Miguel Rodriguez
Antonio Campos Domingo
Pontus Hamberg
Henrik Janhagen
Mischa Alff
Isadora Persson

Albert Cervin
Sakarias Johansson
Joakim “JoCCo” Lindqvist
Igor Khlepitko
Liza Shulyayeva
Caroline Striessnig
Jake Shadle