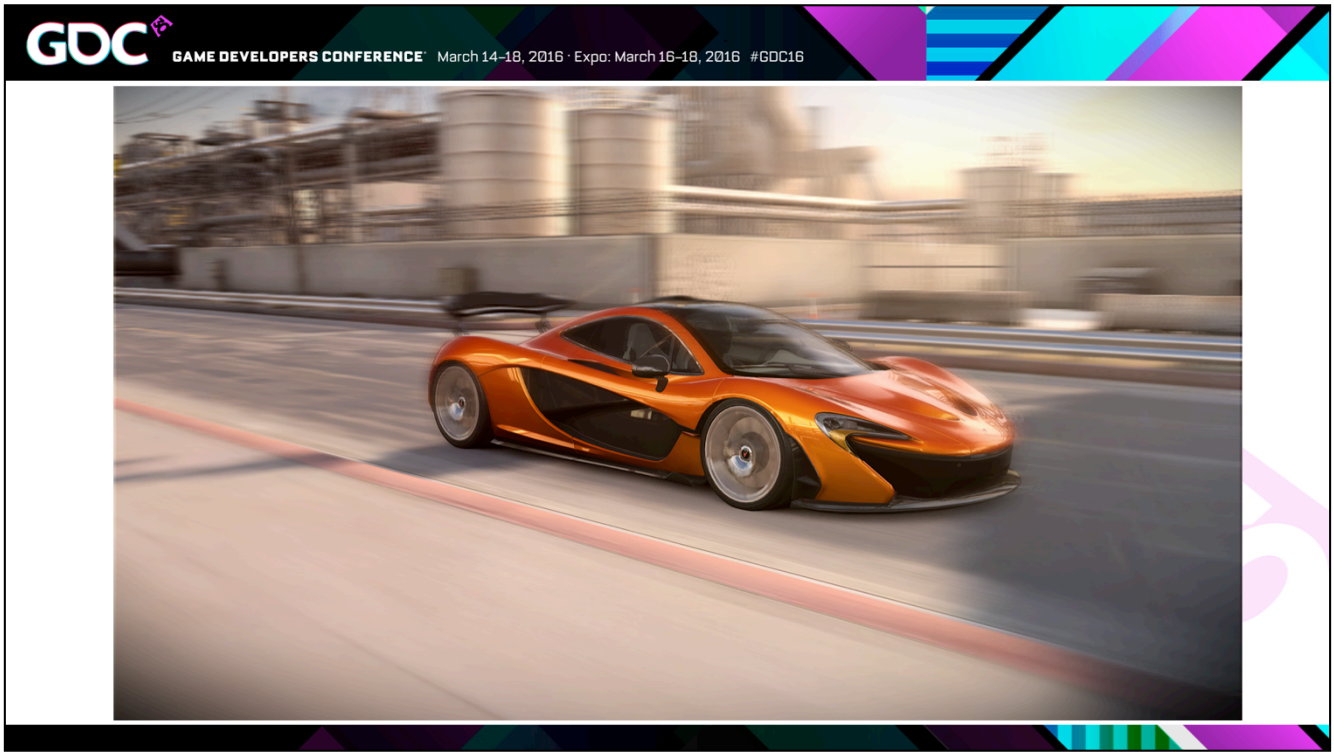

GDC 16
The Environment Pipeline of CSR Racing 2
Scott Harber
Principal Technical Artist
NaturalMotion Games
GAME DEVELOPERS CONFERENCE March 14–18, 2016 · Expo: March 16–18, 2016 #GDC16
UBM



So you've all seen sessions in the past where people show aspirational graphical tech demos to showcase the maximum extents of the hardware?

This session isn't one of those.

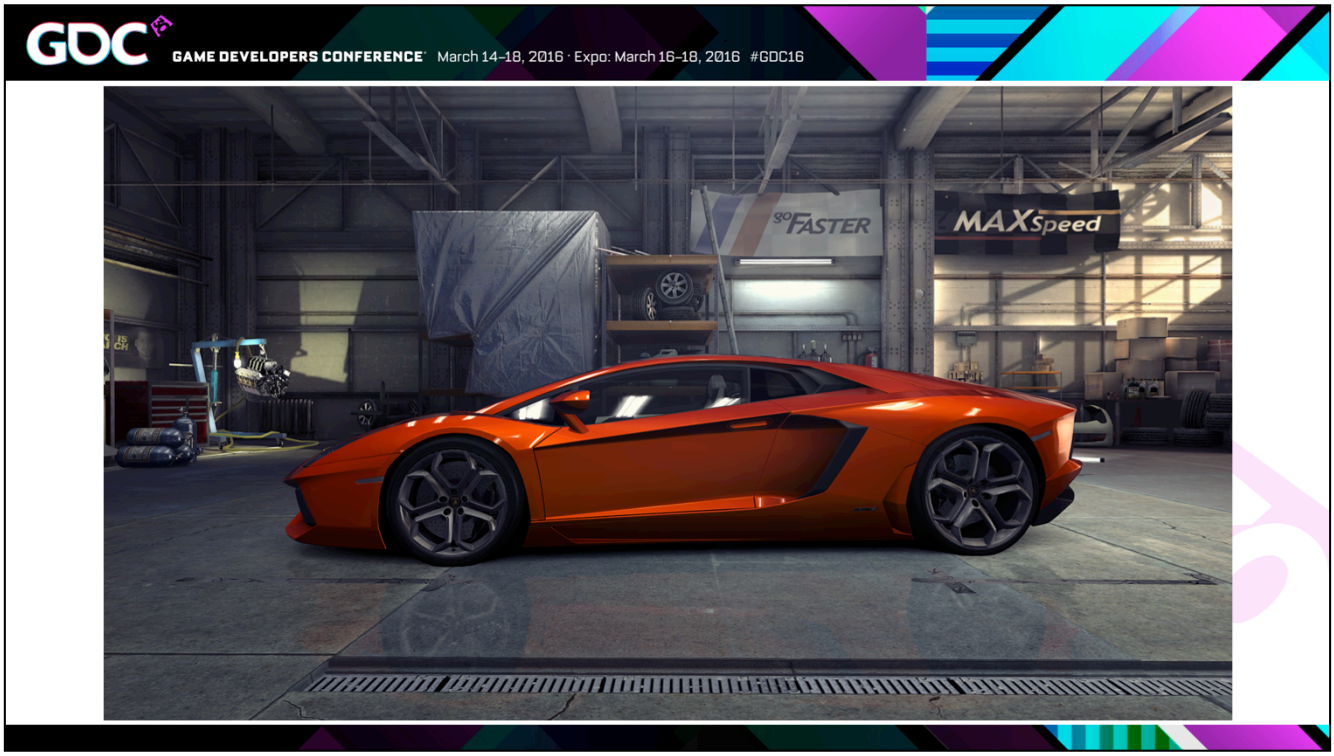
In my experience, tech demos are great, but they're also misleading. I'm more interested in the practical aspects of delivering high end mobile graphics.

Once you factor in gameplay code, UI, networking etc, your available resources for graphics will look very different.

For reasons I'll explain later, this is especially pertinent for mobile.

CSR Racing 2 is used here as a case study.

It’s a drag racing game (made in Unity 4, for iOS and Android).



Based around the development of CSR2's environment pipeline (there is a separate presentation about the vehicle pipeline).

From the outset, we knew we would face challenges in development.

For context, the challenge we faced can be surmised in a single image.

This is the original CSR Racing from 2012

Played by 2% of the world's population – one of the largest install bases of any racing game in existence

We'll talk about its focus on graphics for the benefit of the discussion

At the time, considered some of the best visuals on mobile. CSR2 would have to be a generational leap

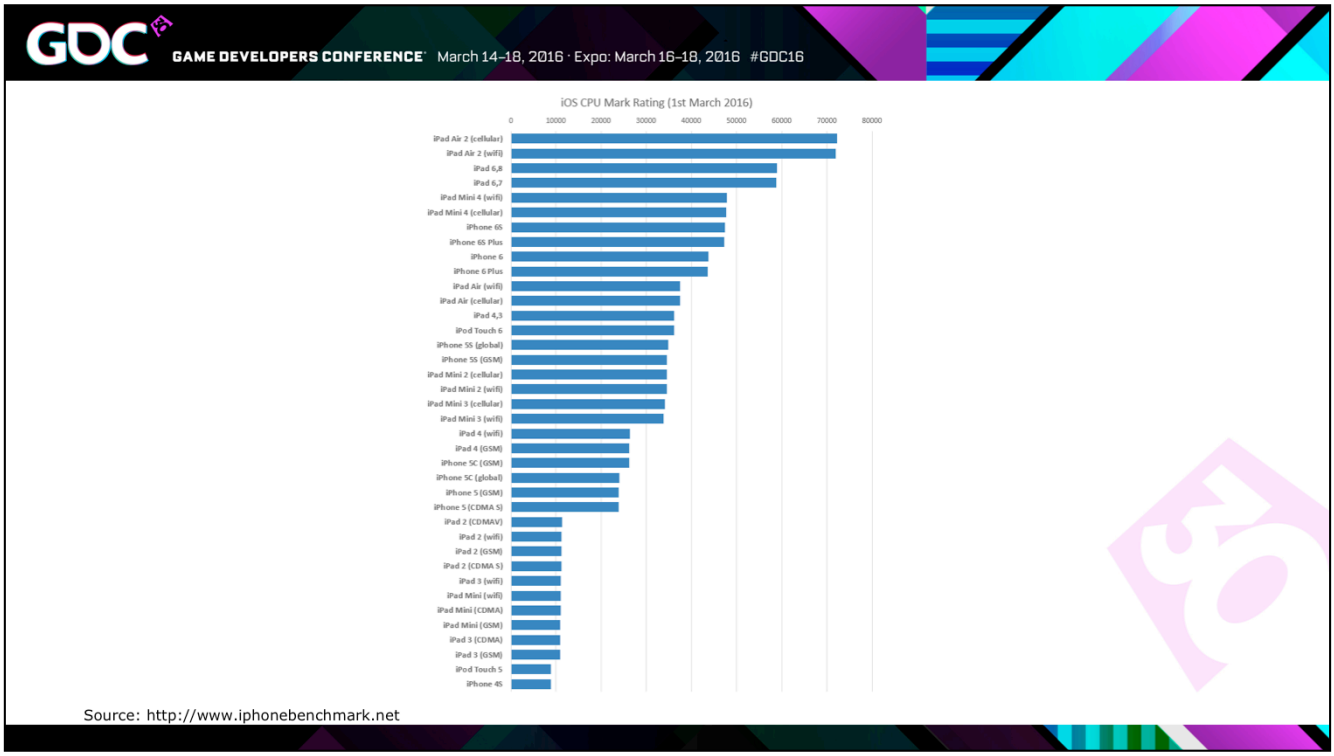
Internal mission statement: "It needs to look like a AAA console game"

Also

The game needs to support every device from the last four years.

That’s 19 iOS devices and roughly 2500 Android devices.

All with extremely varied specs…



Would be the equivalent of making a PS4 game, that also runs on Dreamcast, and supports every console in between.

Environment-Specific Goals

The environments need to support wet weather and night effects…

Plus, we also wanted to support both night time and wet weather variations for our environments

Environment-Specific Goals

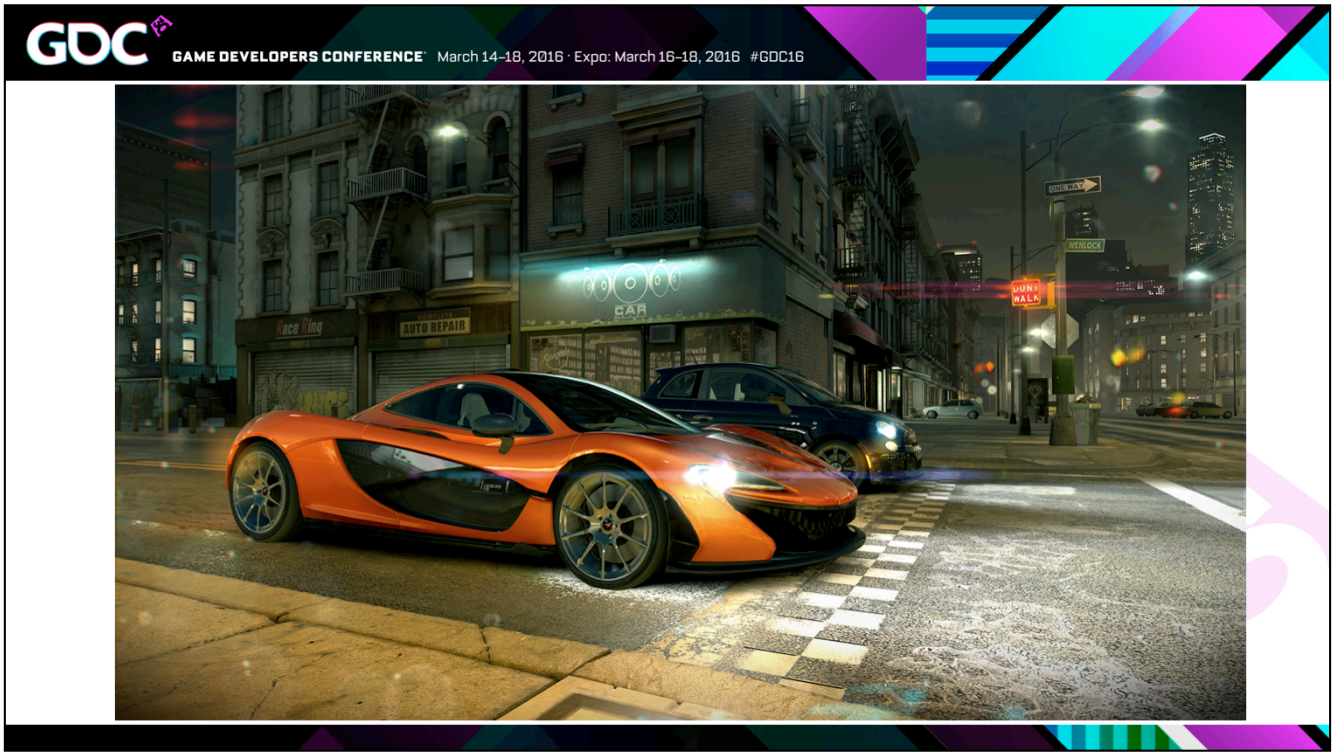
The environments need to support wet weather and night effects…

…and maintain 30fps on all devices

all while maintaining a 30fps frame rate on all devices.

Not to beat around the bush, but we succeeded!

Here's that shot from CSR1 again…



…and a comparable shot from CSR2

But this result was not immediate

In fact, when we first started, our first playable build looked more like this…

It was a deliberate exercise in trying to work within safe budgets.

But while an environment like this would guarantee no performance or memory problems on any device, the visual quality just wasn't there

Going back to the original mission statement: "Is this console quality?" No

Following this, our art director made a paint-over image, using the pre-pro environment.

The result was still below our visual target, but provided a valuable benchmark for the art and tech teams.

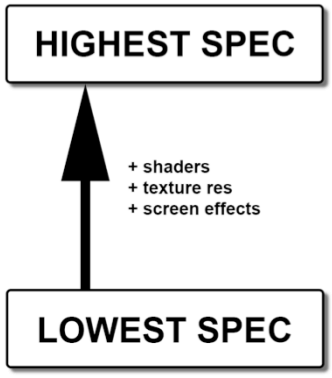
Platform scalability

Need to stop trying to do everything on every device, and trust that the audience is savvy enough to expect that newer devices = better visuals

AAA has had this concept for decades. So far, mobile hasn't

Because of this, there is no immediate frame of reference for how to scale successfully. Need to look to AAA for guidance

The "Bottom Up" Approach

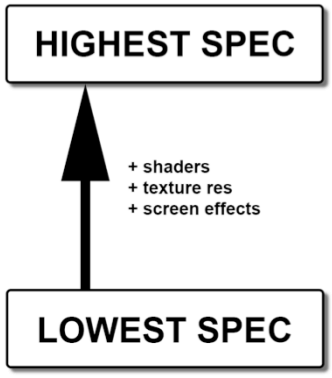


How AAA games were made in the PS3 era.

Core visuals built for low spec, superficial improvements for the high spec (texture res, post FX, shaders etc)

Always limited to the specs of the lowest platform.

The "Bottom Up" Approach



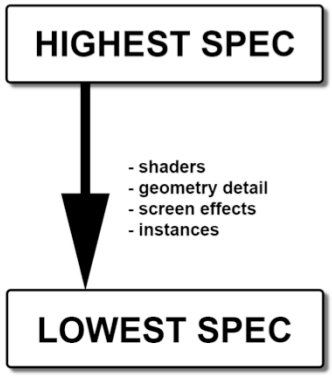
Not really conducive to post launch product paths

Doesn't work well for DLC.

Doesn't work well for the long product paths that mobile is known for.

Doesn't work well for pushing high end quality as far as possible.

The "Top Down" Approach

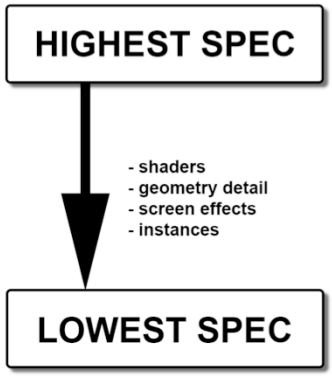


How AAA games are made now.

Build for top spec, and remove features to fit on the lower specs.

The result is less consistent between platforms, but ultimately, higher quality overall.

The "Top Down" Approach

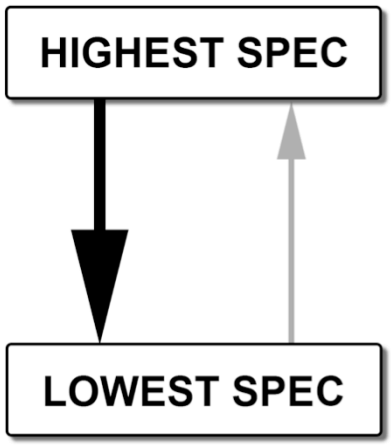


This can be taken too far!

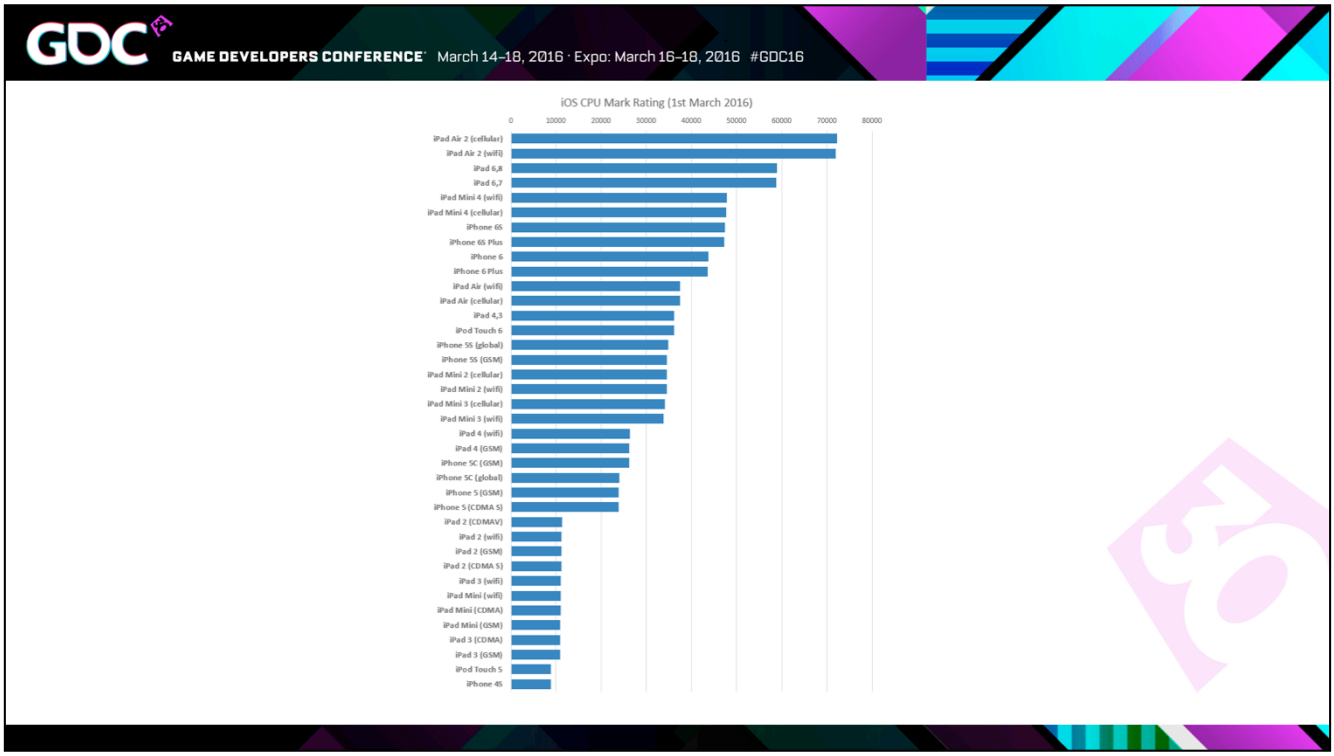
Without naming names, we've all seen comparison videos on Youtube where games have massive quality disparities between their low to high platforms.

We didn't want to do that.

CSR2 is...

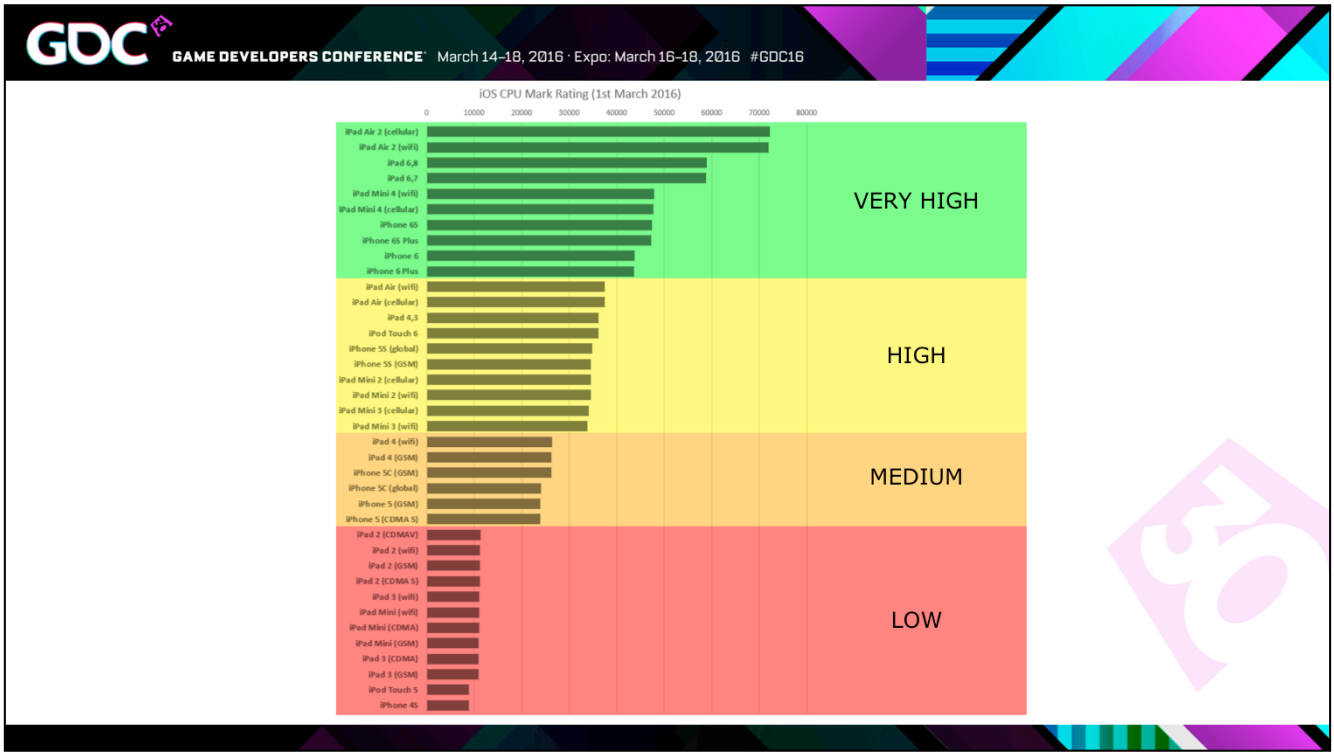


We opted to use a mainly top-down approach, tempered with elements of bottom-up to preserve quality as much as possible.



If we’re going to work this way, we need to build to fewer target devices!

The beauty of mobile is that devices are released annually, and each rollout tends to broadly fall within a discrete range of specs, regardless of manufacturer.



We defined four theoretical quality levels (very high, high, medium, and low).

Treating each one as its own platform, in much the same way as a PC game would do.

The Plan

At this point, we know what we're building to, but not how to build it.

The Plan

1) Use the same data set for all platforms

Didn’t want to keep re-authoring each environment for every quality level.

Conventional wisdom for mobile is to build each environment as one large model to reduce draw calls, but this comes at the cost of poly counts and memory.

We went against that and instanced a lot of the detail. Between this and material sharing, the memory was quite low, even with high poly meshes.

We took a hit on draw calls. Draw calls are more readily scalable than polygons.

The Plan

1) Use the same data set for all platforms

2) Any rendering feature needs to scale gracefully

This one is important, since we're using a top-down approach in a sensible way.

Features must be additive to quality by definition.

If a feature would leave the game looking worse or broken by its absence, then we couldn't use it.

Example: Procedural textures.

Scalable?

- HDR
- Real-time lighting/shadows
- Deferred rendering

With this in mind, we knew some features would have obvious problems with platform scalability.

So we chose not to investigate them.

How that looks in practice…

"Low"

All post effects and specular disabled

30 draw calls, 20MB memory

"Medium"

colour grading and dynamic exposure enabled

Higher res geometry

200 draw calls, 100MB memory

"High"

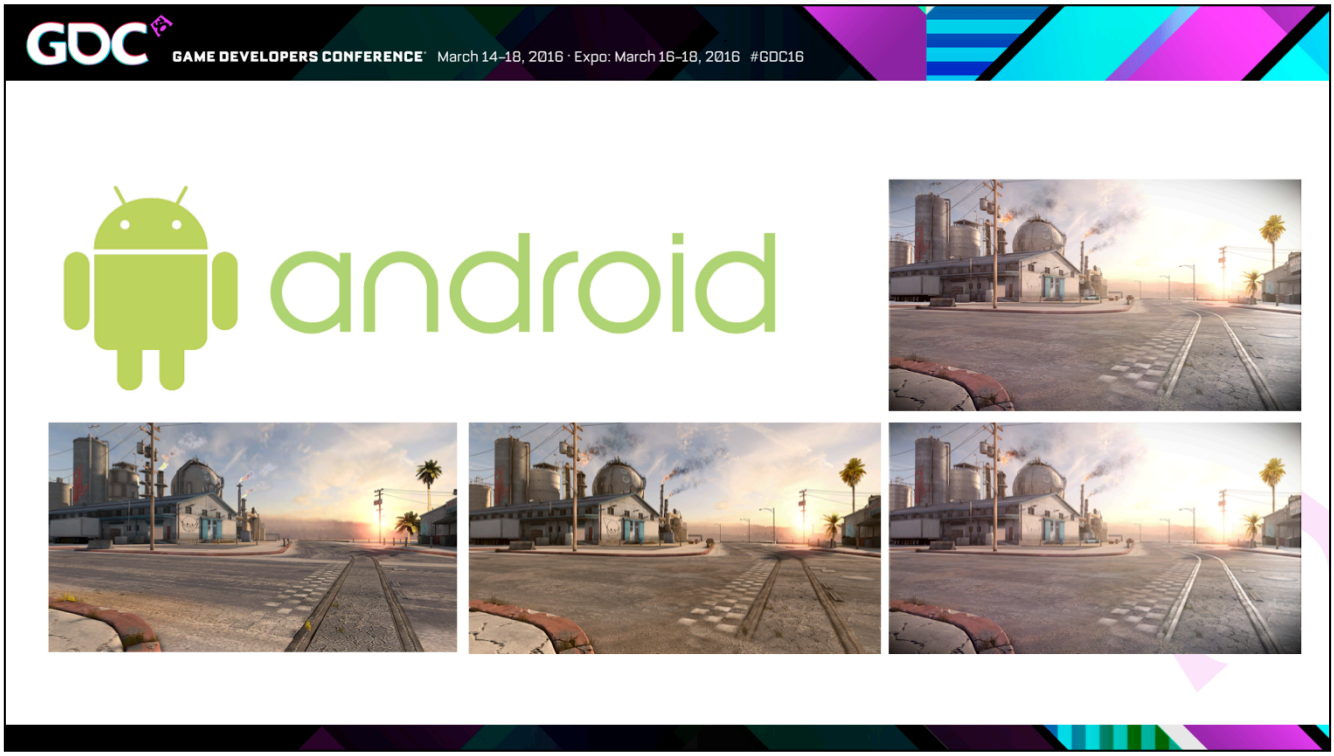
Physically based specular enabled.

Post effects enabled.

300 draw calls.

"Very High"

As before, but with more accurate fresnel and motion blur



I've included this slide to illustrate a point about multi-platform development.

While different devices may fit within the same quality level, comparable hardware doesn't always mean the same hardware.

Due to differences in thermal throttling and optimisation of graphics drivers, Android has very different authoring requirements to iOS. Even when working to the theoretical quality levels, some low level optimisation will always be needed.

In this case, we didn't have as much CPU to work with as iOS, but we had more memory.

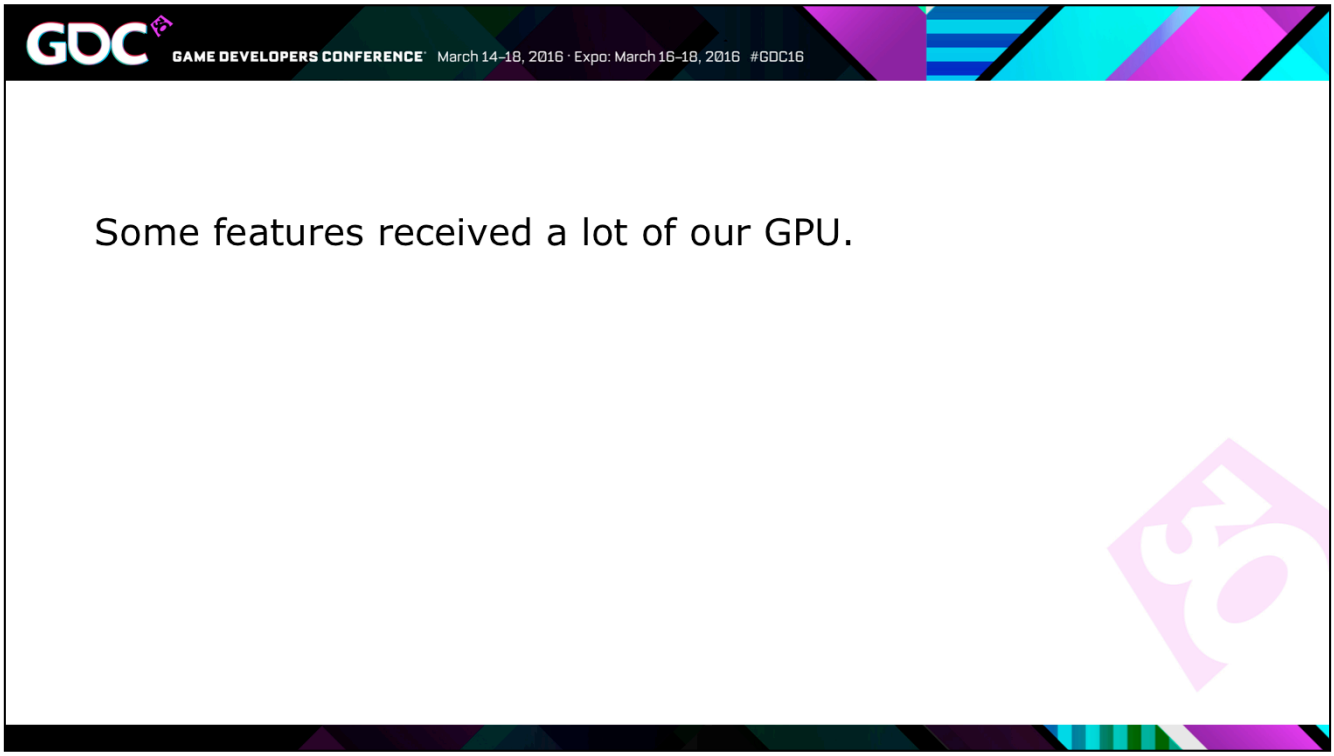
Wrote custom static batching pipeline into our export process. Reduced draw calls with an acceptable memory increase. Got to the same result, by playing to the strengths of the hardware.

Ready for Production!

At this point, we were finally ready for the artists to work.

Building pipeline as they work, to ensure that the pipelines are genuinely useful.

Found that any feature would fit within three categories.



Some features are very important, and require a lot of GPU.

Some features received a lot of our GPU.

Other features were emulated to provide a desired effect.

Others are still important. But to do everything you want to, you need to find more creative solutions.

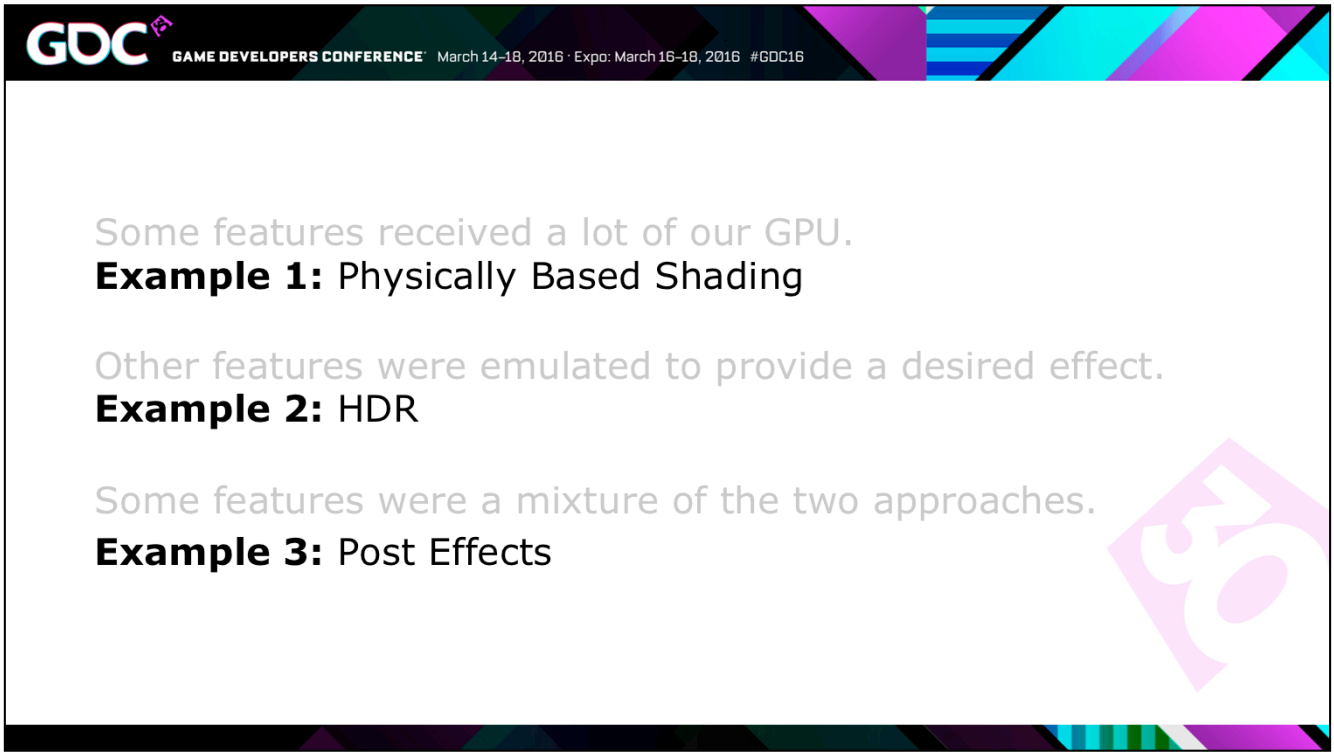
Some features received a lot of our GPU.

Other features were emulated to provide a desired effect.

Some features were a mixture of the two approaches.

Some features are a bit of both.

Rendering spend defined by priority.



Going to cover one example of each

Example 1: Physically Based Shading

Completely bespoke solution. Evaluated Unity 5's solution, but didn't use it.

Mostly inherited from our vehicle pipeline; with the additions of real time directional specular for the sun, lightmaps for diffuse lighting, etc.

The fundamentals of PBS have been covered in many other presentations, so I won't do it here.

New material pipeline had immediate value.

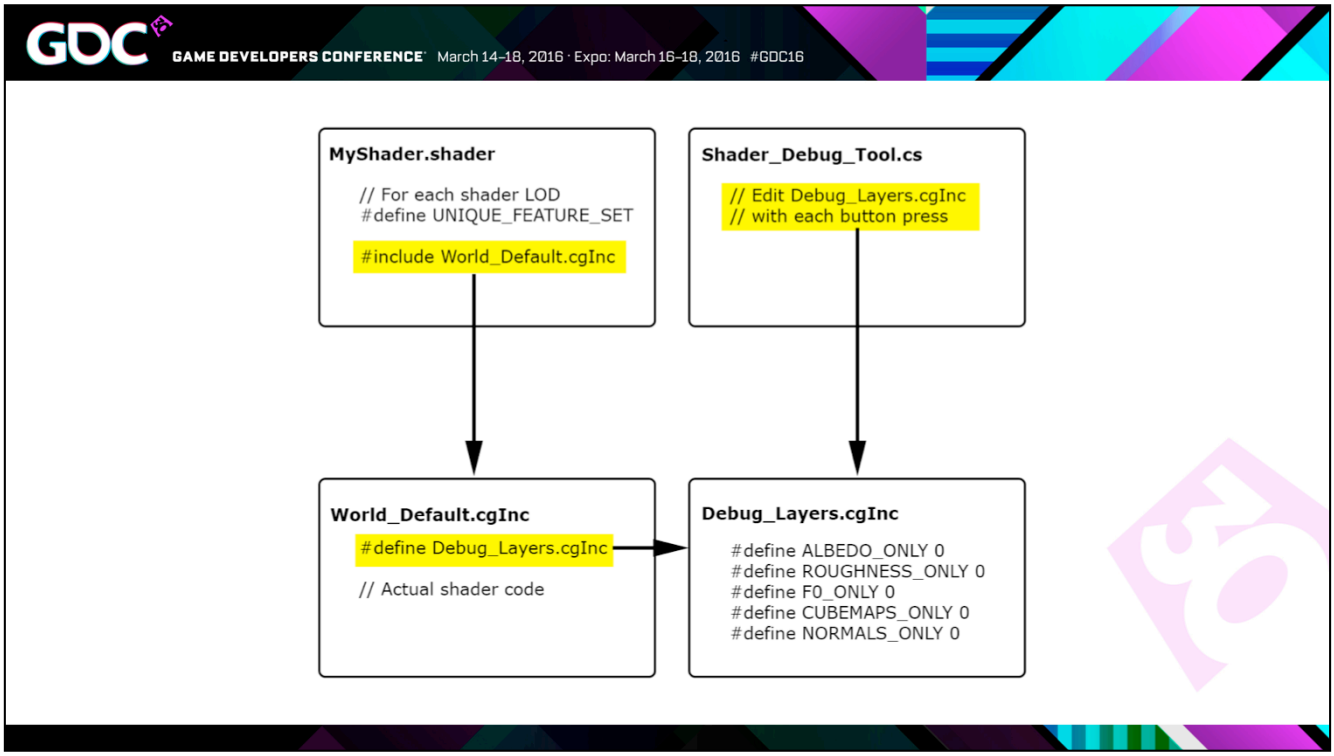
Allowed us to use both authored cubemaps and real life HDR images as testbed environments. We could use these to define material libraries and test incoming assets from the outsourcers.

Workflow – Shader Debug Tool

Typically, features beget workflow. It transpires that writing your own renderer necessitates custom tools!

Specifically, the artists needed a way to view and analyse the various layers of their materials. Also, they needed to be able to preview the effects of the device scalability.

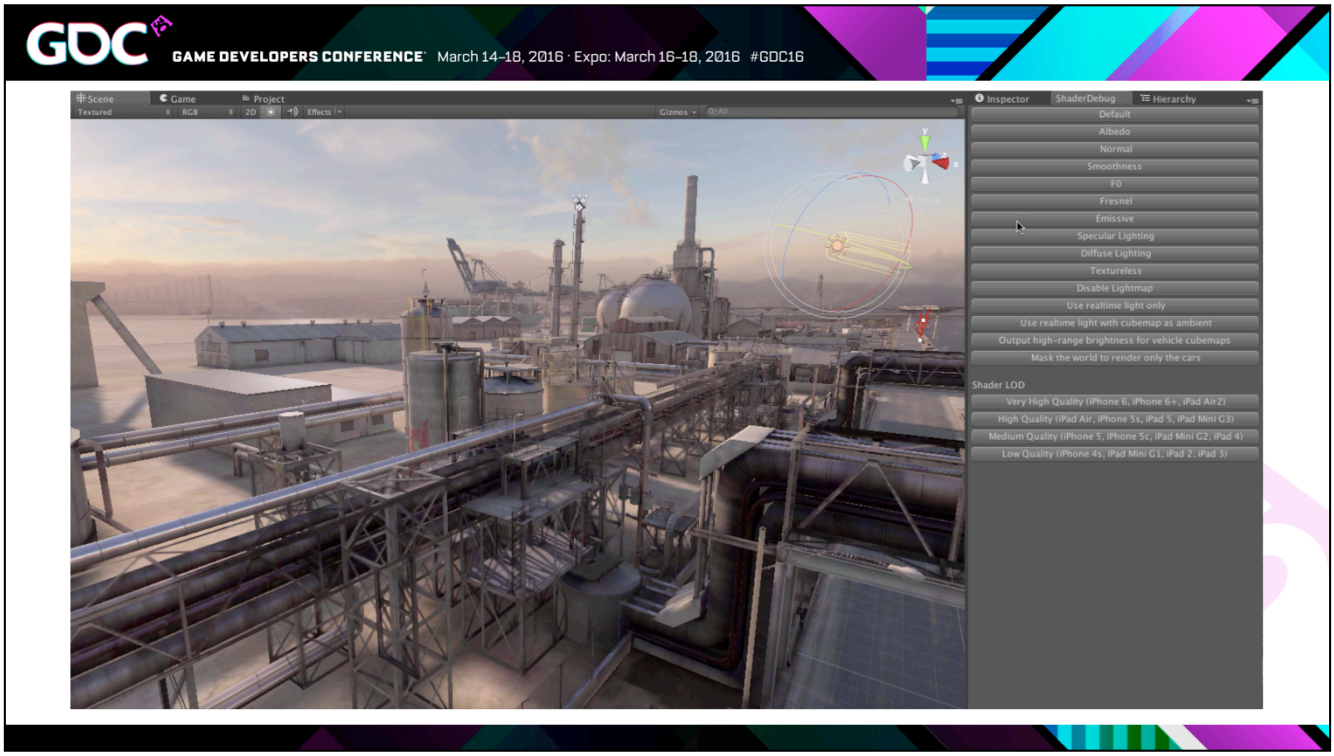
Not forward rendered. We wrote this pipeline, that dynamically edits shaders



Each shader would reference a single uber-shader that controlled everything.

This uber-shader would reference a separate file, that stored preprocessor directives ("#defines") for each rendering layer. When one #define was enabled, it would enable this rendering layer for all shaders simultaneously.

The Shader Debug Tool would directly edit this file, and enable the #defines based on the artists' input.


GDC 16
GAME DEVELOPERS CONFERENCE March 14-18, 2016 · Expo: March 16-18, 2016 #GDC16
Scene
Game
Project
Inspector
ShaderDebug
Hierarchy
Default
Albedo
Normal
Smoothness
F0
Fresnel
Emissive
Specular Lighting
Diffuse Lighting
Textureless
Disable Lightmap
Use realtime light only
Use realtime light with cubemap as ambient
Output high-range brightness for vehicle cubemaps
Mask the world to render only the cars
Shader LOD
Very High Quality (iPhone 6, iPhone 6+, iPad Air2)
High Quality (iPad Air, iPhone 5s, iPad 5, iPad Mini G3)
Medium Quality (iPhone 5, iPhone 5c, iPad Mini G2, iPad 4)
Low Quality (iPhone 4s, iPad Mini G1, iPad 2, iPad 3)

VIDEO

Example 2: HDR

Our approach to HDR was one of the key features that we had to approximate.

We knew from the outset that a true HDR pipeline would have problems with scalability, but we still needed the effect.

It can be done on high end devices. But as discussed, we also need to support the lower devices as much as possible.

Plus, even for the high spec platforms, we had a lot of features to support.

Exposure Adaptation:

We wanted the exposure to change as the player drove along the track.

Because we have an LDR renderer, a white piece of paper is considered as bright as the sun.

Separate the cars, world, and sky; affecting their exposures independently of each other.

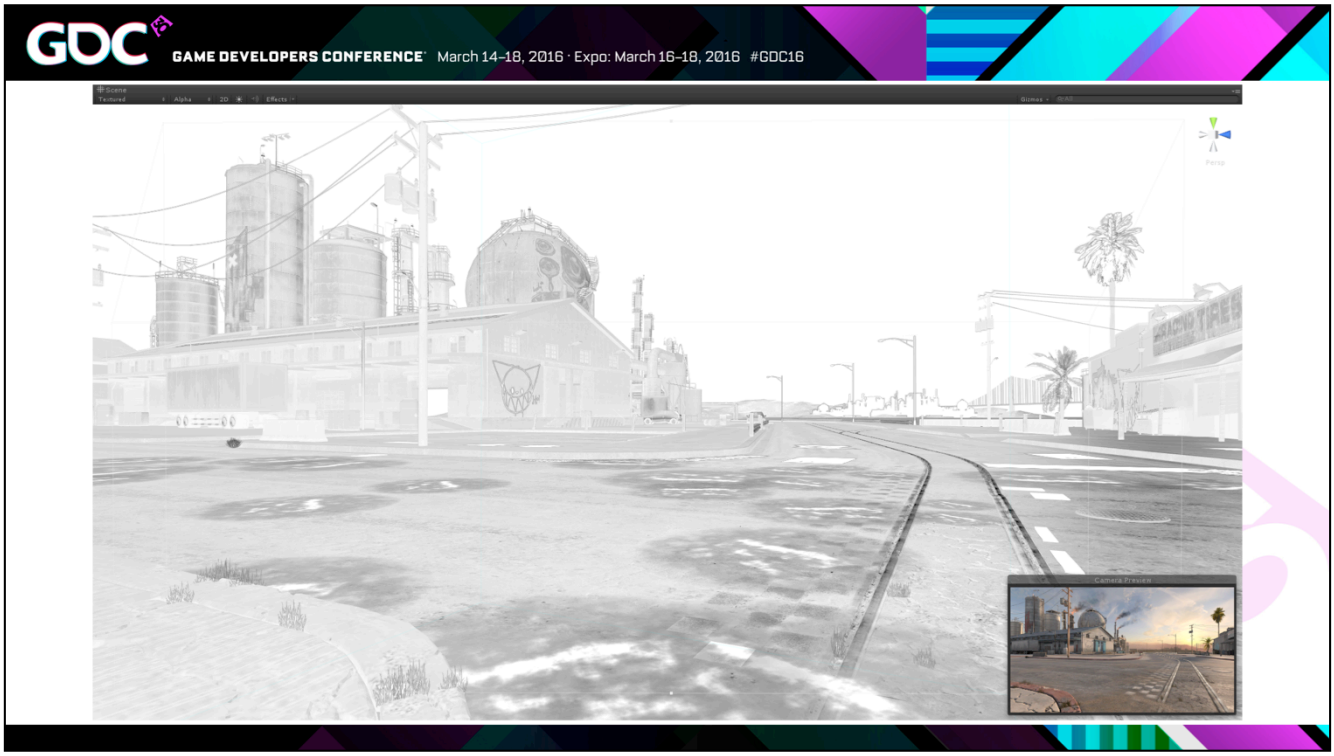
These variables would be affected by artist-authored trigger boxes that were placed along the track.

Bloom:

Basically the same problem as the exposure adaptation.

How do we avoid affecting the bloom range of any white objects in the scene?

The three-layer exposure approach wouldn’t work here, as (in the above example), what if we only wanted to affect some of the environment instead of all of it?



We were able to recycle the alpha of the frame buffer as mask for bloom effects

In the bloom shader…

```
result *= 1-tex2D(_MainTex, f.uv).aaaa;
```

Relatively cheap line of shader code that is very powerful.

Here is a shot without any bloom

If we wanted strong glints on the cars, we can mask the bloom to only affect the extremes of the vehicle specular.

If we wanted an over-exposed sky, we can mask the bloom to only affect the sky.

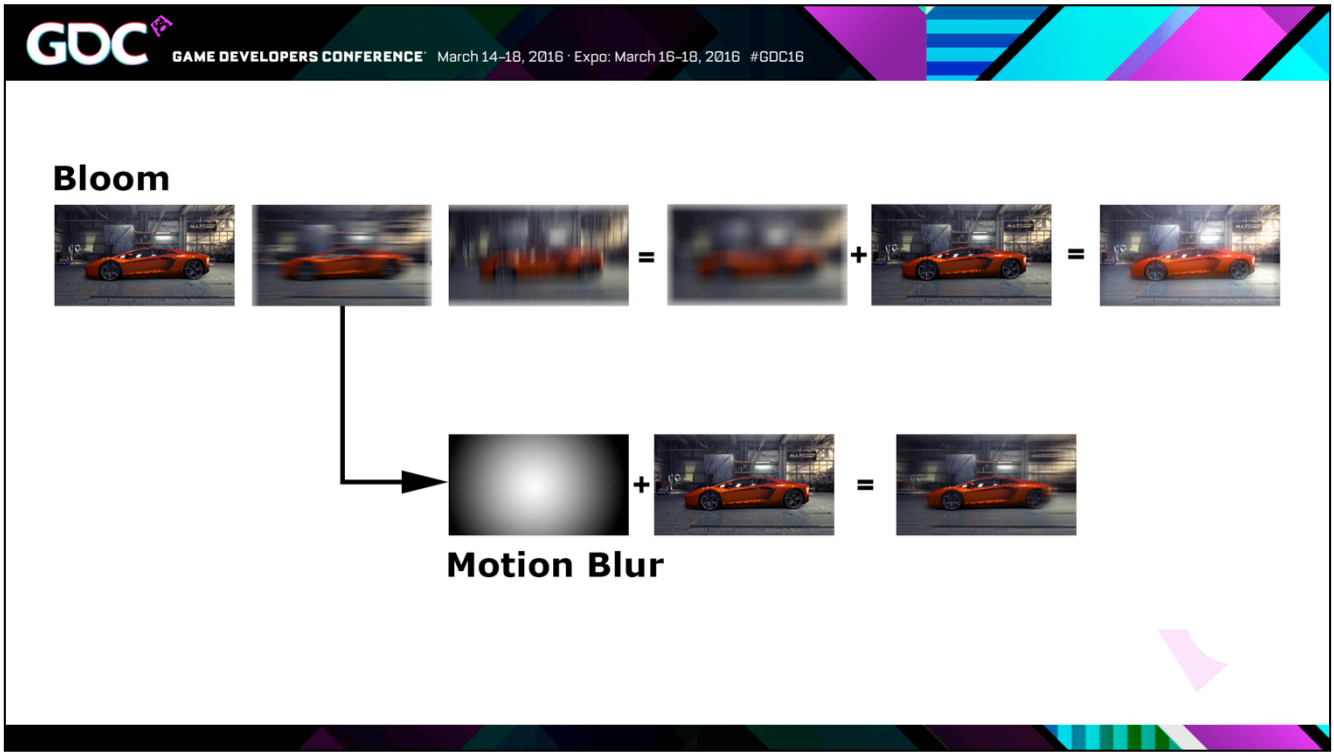
What if we wanted to do volumetric lighting, where any emissive elements appear to light the surrounding fog?

For this, we can simply mask the bloom to the emissive materials, then reduce that mask, based on the distance of the fog.

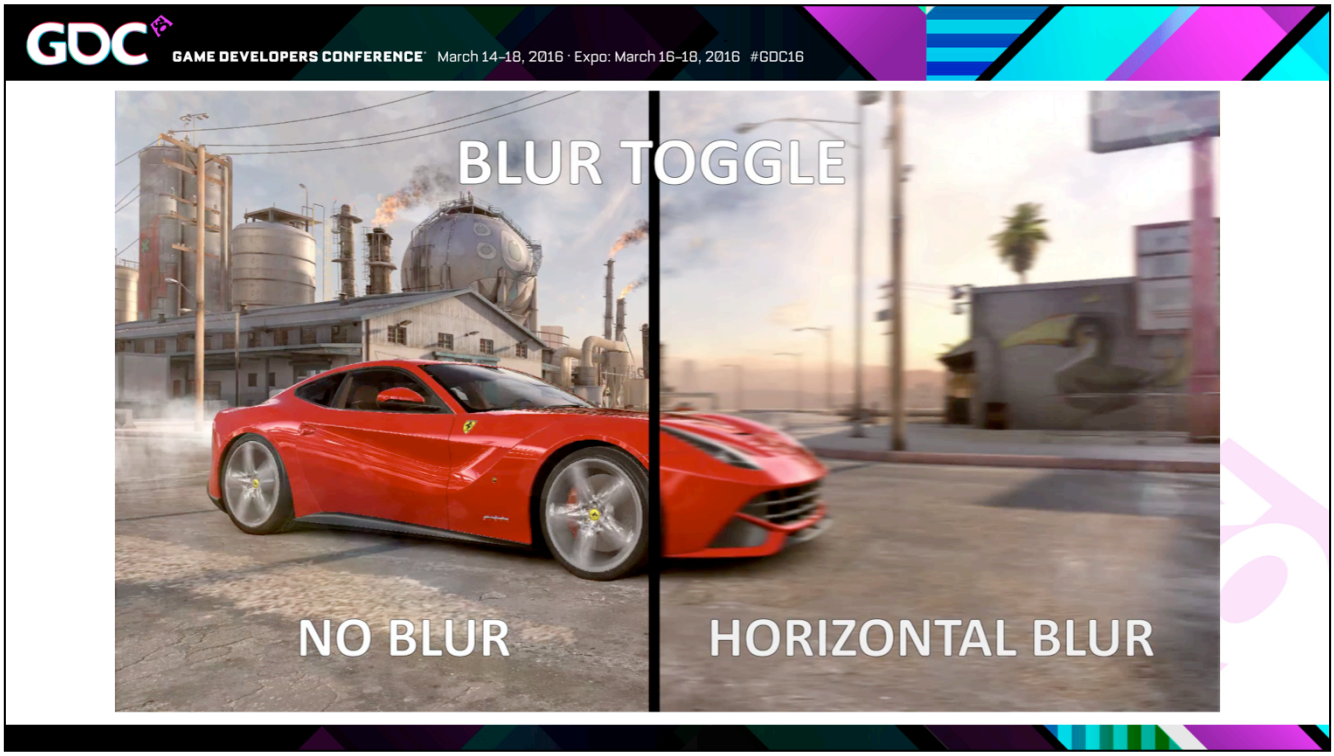
Like this

Example 3: Post Effects

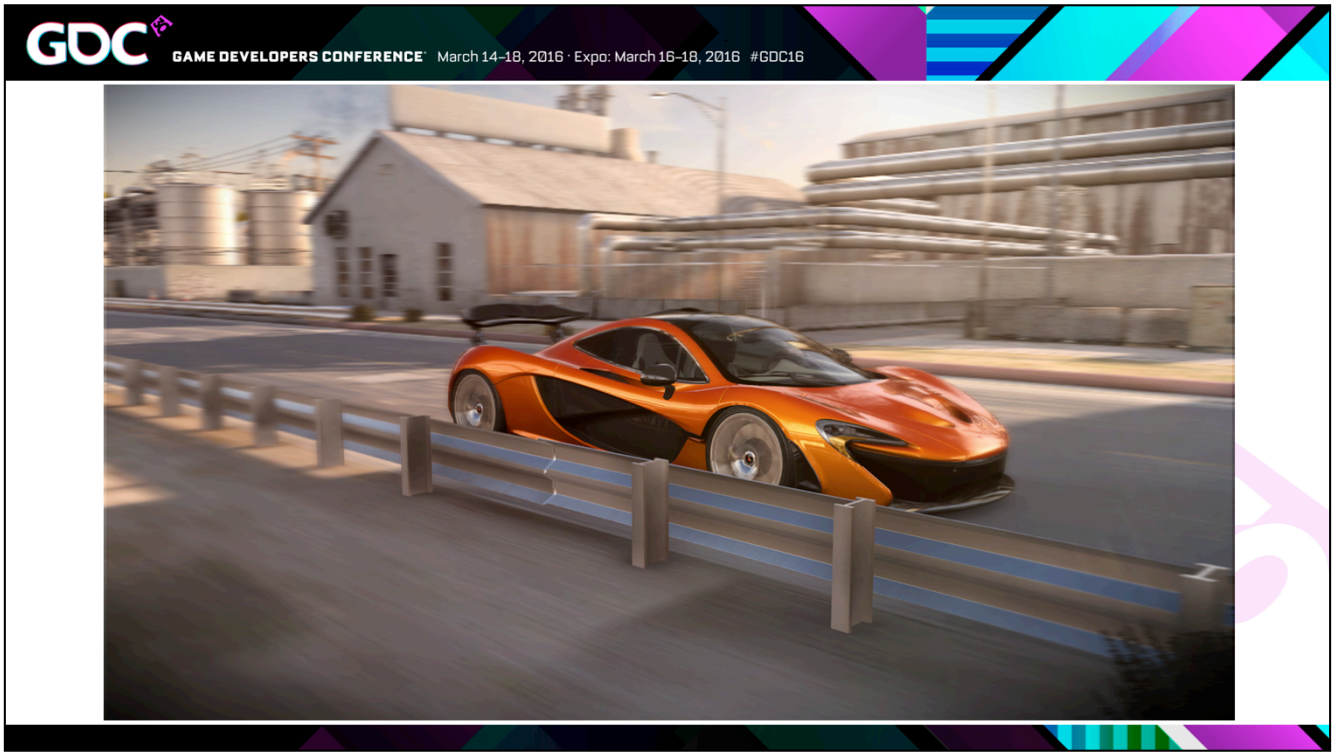
Because of our previous decisions on the renderer, we were able to improve and add to the post effects pipeline by leveraging our existing systems.



Out initial motion blur: Recycled blur from bloom shader



Vignette renderer recycled into a separate effect



Which gives us this result (as released on high spec devices)

Later replaced with a gaussian blur on very high end devices

Blur the bloom at a different rate for effect of HDR motion blur.

Four features for the development cost of one!

About those weather and night effects…

Most of the other features for post effects were borne from supporting weather and night effects

Wet weather became our showcase for everything we can do in the post FX pipeline.

Defined a lot of requirements for what we would need to support.

We knew we'd need colour grading, and some full screen effects (rain on lens)

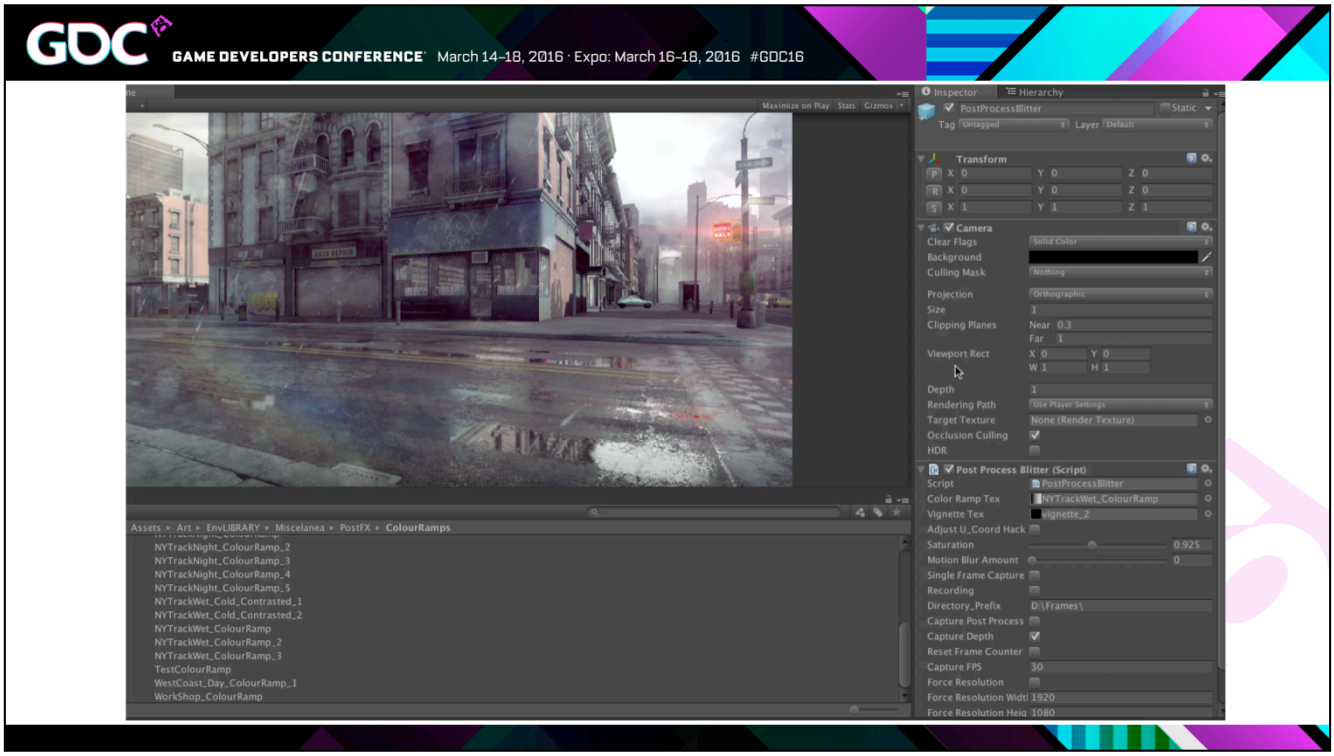
Wet roads are a very simple thing to do with physically based shaders.

Heavy fog, height fog, soft flares, sky bloom

Night time; basically the same elements.

Change window illumination, change shape of lens flares etc

Through the use of various global scope shader constants, we could create very different scenes, without changing the scene or materials



Post Effects:

- Grade
- Saturation
- Motion blur (and lens effects)

Specular features:

- LDR scale
- LDR offset
- Real time sun scale
- Wet road
- Night time window illumination

Material level effects:

- Height fog
- Impact splashes
- Vignette

Post effects:

- Bloom
- Sky bloom

In conclusion

Feature scalability!

Feature scalability per device is essential to achieving the highest possible visual quality

In conclusion

Feature scalability!

Expect to be “creative” with some features.

Expect to be creative with some features. I remember a GDC presentation from 2011: “in three years, mobile devices will be as powerful as Xbox 360s”. This happened, but as I mentioned before, comparable hardware is not the same hardware. You’ll always be looking for ways to get the most of the mobile hardware.

In conclusion

Feature scalability!

Expect to be “creative” with some features.

Think about the future.

Give some thought about how to best support future devices that release after your game does.

For example, the iPad Pro came out after we soft launched, and it doubled the specs that even our highest devices previously supported. The question is “how to spend this extra GPU”.

Top-down development in its purest form won’t work here; you can’t delete features on an already-existing product, just to support something new.

Bottom-up development will only provide superficial improvements.

- If you like what you've seen and are interested in being part of the great work we're doing, come see our recruitment team as we're hiring.