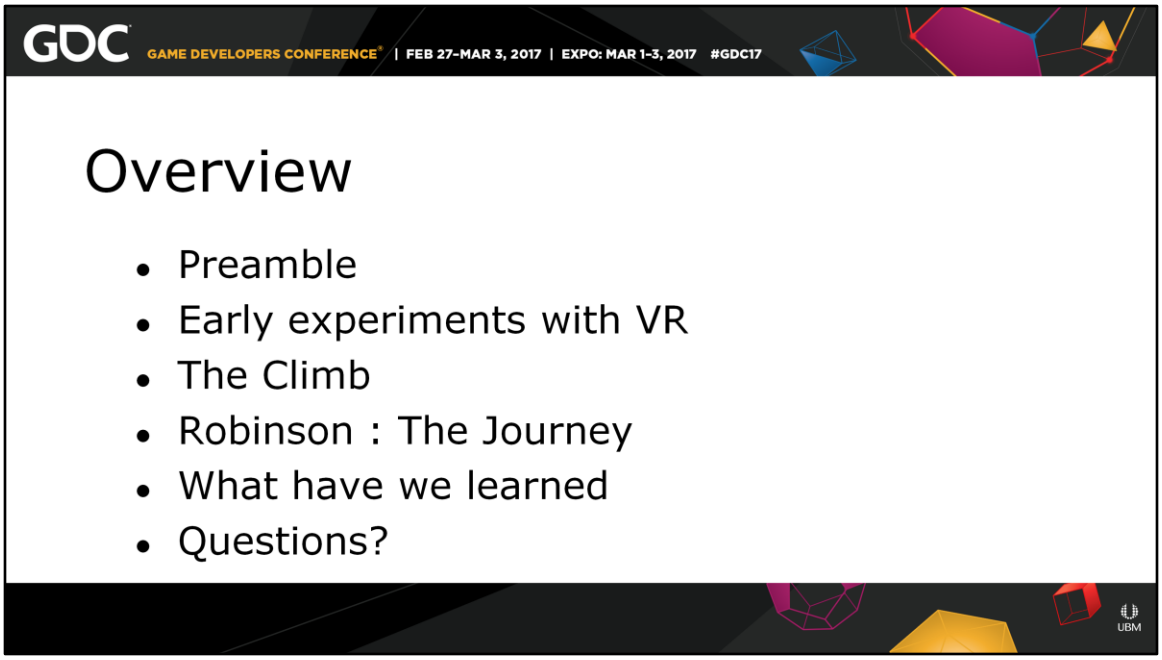
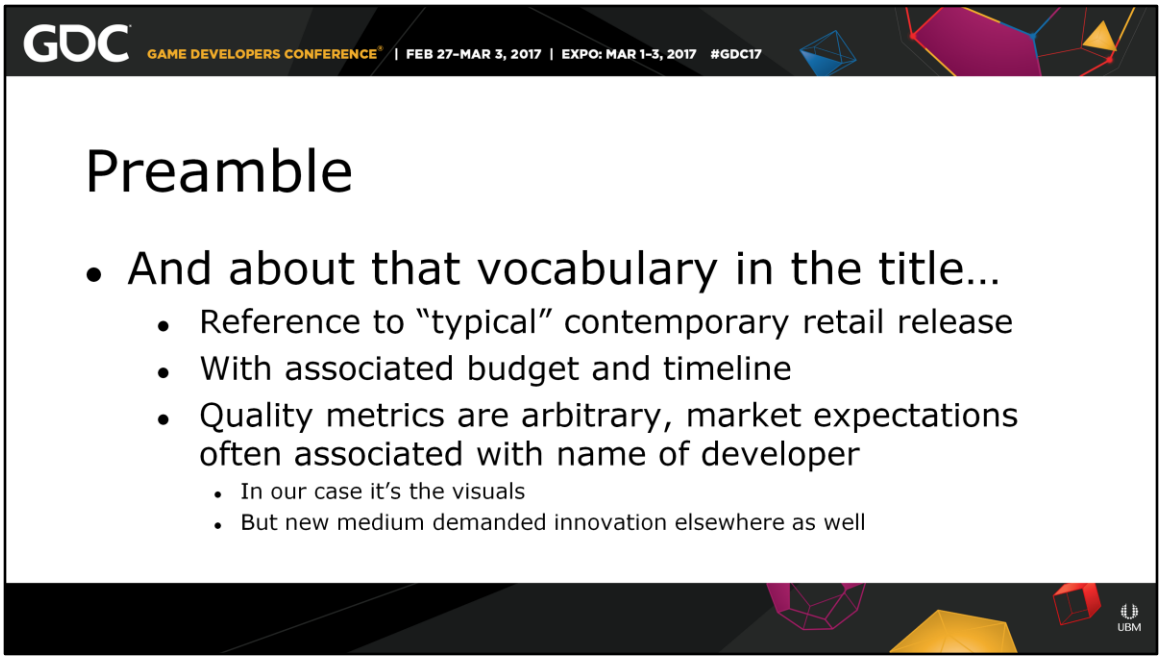
Hi, I'm Rok Erjavec, and today I'll be talking to you about VR, and the associated challenges of developing against market expectations from a 'AAA' developer, while working with the market and budgetary constraints of this new medium.



Postmortems of Climb/Robinson will do a bit of a dive into some of the tech-successes/innovations we delivered during this period as well.

2016 was a year of many firsts for Crytek:
First VR release, and consequently first in-house PS4 (and PSVR/Oculus) releases, first game with no weapons (2x), first commercial game with Dinosaurs (given the history, a bit of a surprise), first console release from Crytek to run at 60fps and many more.
Last and not least – the experience of going through shipping 2 full-priced launch titles on a new medium on two separate launch platforms, from conception to release – in just a bit over a year, is one that no one in the teams would have predicted when we started all this.
Given all that – we felt it would be worth sharing the experiences of doing all these firsts while simultaneously trying to hit market-expectations on some very constrained timelines and with limited resources.

Obviously this also coincides with many things in 2016 being game industry firsts – and VR as a medium creates so many new unknowns, that just talking about how we dealt with them on first try is a learning experience into itself.



Also – before moving forward – lets clarify the vocabulary of the talk-title used – "AAA" being rather poorly defined as a term. Today we'll be using it to as reference to contemporary market expectations in a (usually full-priced) retail title, developed on a reasonably standard 2-3 year schedule with development budget in 10s of m$. Not referencing any particular quality metric(s) here – and market expectations will obviously also vary even just by the name associated with the product – in our case, it was clear that visual quality has to live up to the reputation.
That being said – being in a new medium, there was inherent expectation of innovating in other ways…

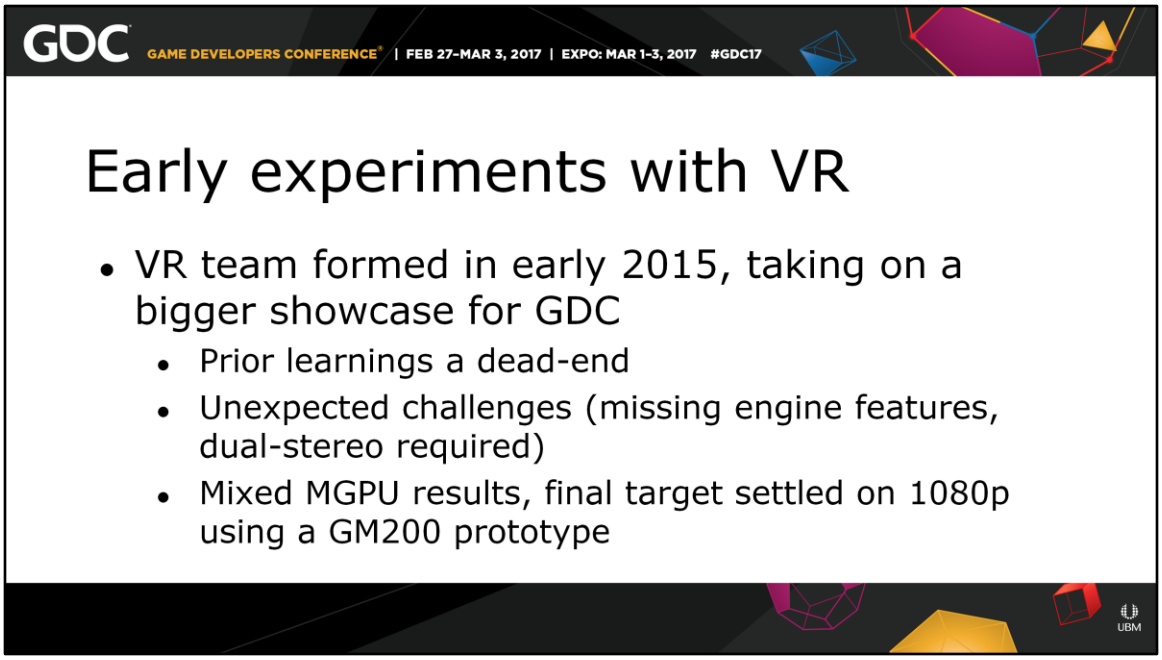
Finally I'd mention with regards to the vocabulary – if I slip during the talk and refer to "2d" games, I'm referring to non-VR flat-screen experiences, not actual 2d visuals.

Early experiments with VR

- Crytek first foray into VR in second half of 2014
  - Use existing assets + stereo (reprojected) = profit?
  - Mixed results (performance and quality) but promising response in early showings

First VR tests in CE were made in second half of 2014, with a short demo using existing Crysis assets and a quickly thrown together rendering path that used reprojected stereo from our previous titles.

Early showings garnered promising response from general public – but it wouldn't be until a few months later that we learned just how mixed the results were.



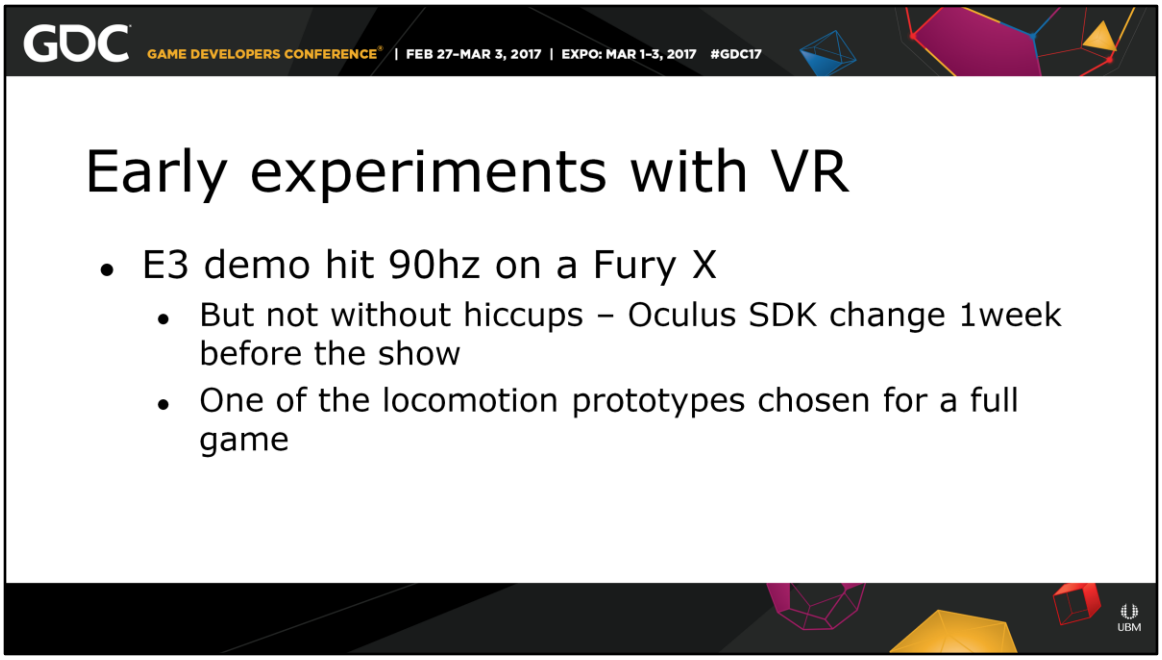
Early learnings turned out to be flawed – Stereo reprojection did not even approach the expectations and metrics for VR set by Oculus, and in addition to this – some fundamental missing pieces were identified (such as lack of support for asymmetric FOV in the renderer) that further impacted correctness of the early VR-rendering code-path.

Thus GDC showcase turned out to be a bigger challenge than anticipated. Hitting the 90hz target, especially once we discovered dual-rendering stereo could not be avoided and having to hack missing engine features (support for asymmetric FOVs) led to a mad-dash to complete demo in time for GDC.
Last minute experiments with MGPU did not yield sufficient improvement (engine at the time was ill-suited to effectively leverage this), so last minute change was made to run on a prototype version of latest NVidia cards. Resolution for the demo was set to 1920x1080 – which was deemed to be sufficiently high for the fidelity on display (and as we would later see – public showings would never pick up on that fact).

Early experiments with VR

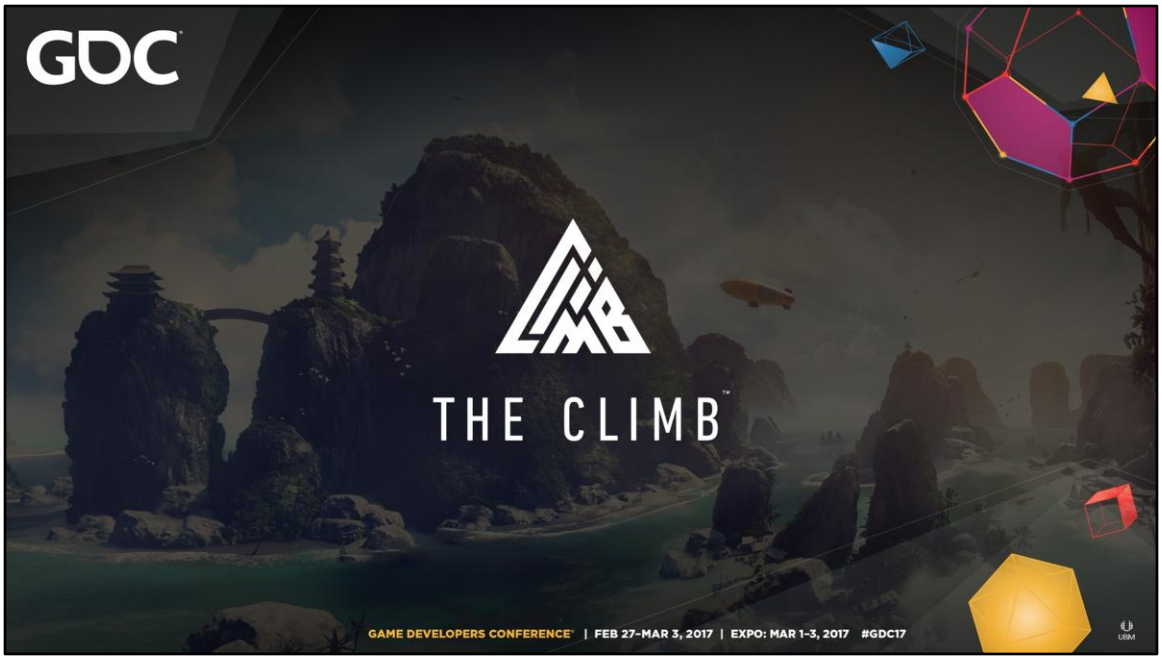
- E3 2015
  - From small stationary scale@GDC to an "open" level
  - Early artificial locomotion prototype
  - First engine code-path "optimized" for dual-stereo&VR

GDC demo was a stationary, "room-scale" experience with minimal interaction (only head-tracking used as input).
E3 ambitions were to move into a larger area, with an actual "open" environmental traversal, with more environment interactions and actual "gameplay" elements.
This, along with other ambitions, led to starting prototyping locomotion in VR, going through the usual set of failed experiments most people go through when they first try artificial-locomotion in the medium – VR test-station being labeled Barf-station at some point aptly summarizing the success of most of the early prototypes.
In parallel – engine work was on-going to streamline the largely un-optimized hacks that were used to run at GDC – removing redundant work (multiple shadow passes, unnecessary post-work), improving stereo-submit pipeline, making temporal-AA work with VR and more.



The improvements in question led to E3 demo actually bringing enough performance improvement for it to hit 90 on more normal GPUs (shown on FuryX, but also hitting targets on GTX 980, and being in striking distance on 970s), even with the much larger scale of the level. Resolution was kept at 1920x1080, as improvement to AA pipeline resulted in image-quality that everyone at the time was fairly happy with.
Of course – not everything went smoothly - last minute requirement to adopt a newer Oculus SDK would lead to a non trivial amount of last-minute work (V0.6 was a major change in interfaces too).

Locomotion prototyping eventually yielded some models that were deemed usable, and one in particular stood out, using combination of head-tracking and “pulling the world” with button presses. A variant of the same model was used for the E3 demo itself.


GDC
THE CLIMB™
GAME DEVELOPERS CONFERENCE® | FEB 27–MAR 3, 2017 | EXPO: MAR 1–3, 2017 #GDC17

The Climb

- Prototype -> game pitch -> contract
  - Preferred locomotion prototype chosen as the core mechanic.
  - Use the core game-loop and iterate/refine, avoid growing the scope

Shortly after E3 demo, The Climb was officially started as a project. Climbing/Grabbing locomotion that really shined during prototyping was chosen as the core mechanics, and we've put together a pitch for a complete game around it.

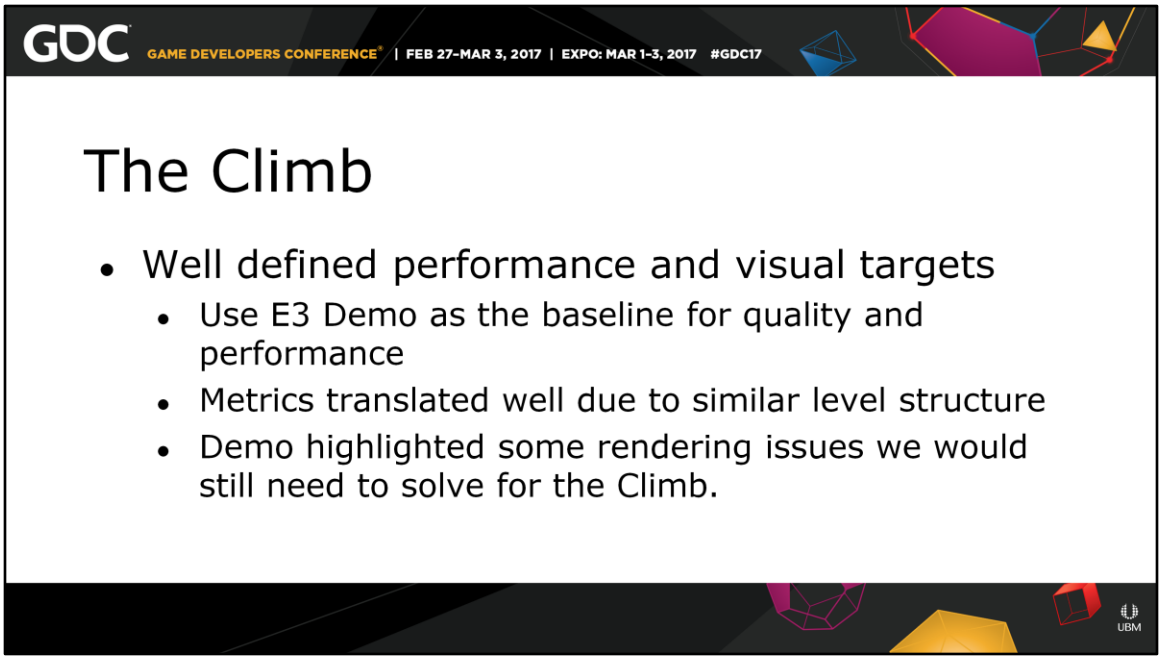
The objective was simple – maximize the core-gameplay loop through further prototyping and refinement, but keep the scope controlled and focus any additional expansion of the content built on top of the core-gameplay loop, never along-side it (eg. collectibles, challenges, achievements, multiplayer).

The Climb

- Aggressive timeline, and delivery plan
  - Initial pitch called for a 10month dev-cycle
  - Milestones every 4-6weeks
  - Worked as sanity check against scope expansion
  - Also safety against unknowns of new hw/sw platform

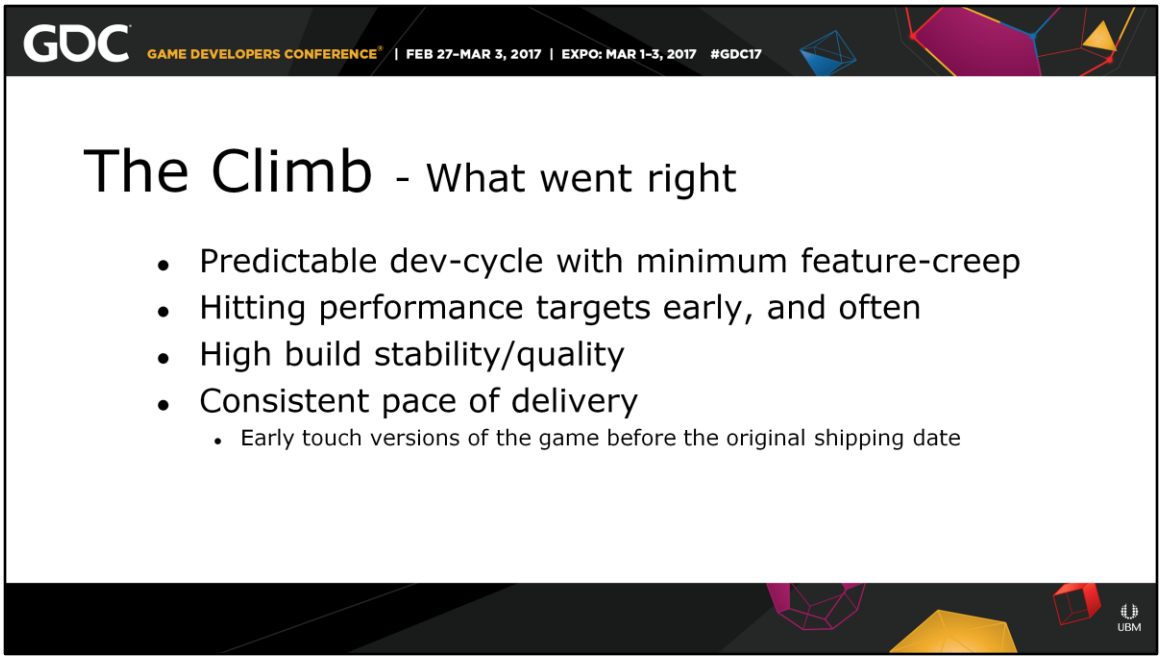
Initial pitch called for 10months development cycle, detailed in what was basically a waterfall plan for entire development, with strict deliverable milestones every 4-6 weeks. This would turn out to be helpful for keeping scope-under control, as any changes/additions would always have to be sanity checked against the next coming list of deliverables.

Additionally it would be used as a shield (and arguing stick) against the unpredictability of working with a completely unknown (and at the time, very incomplete) hardware AND software platform.



Performance and graphics targets

E3 demo proved very valuable as a baseline to set art quality and performance benchmarks. Structure of the level was loosely similar to what we wanted in The Climb (ascending walls, large open-landscapes, emphasis on surface detail and ambient life), and NVidia 970 being highly rumored as the recommended target for Oculus. Additionally – some of the rendering issues demo never solved (such as high distance shadows – vista extending up to 50km) would be applicable to The Climb levels as well.



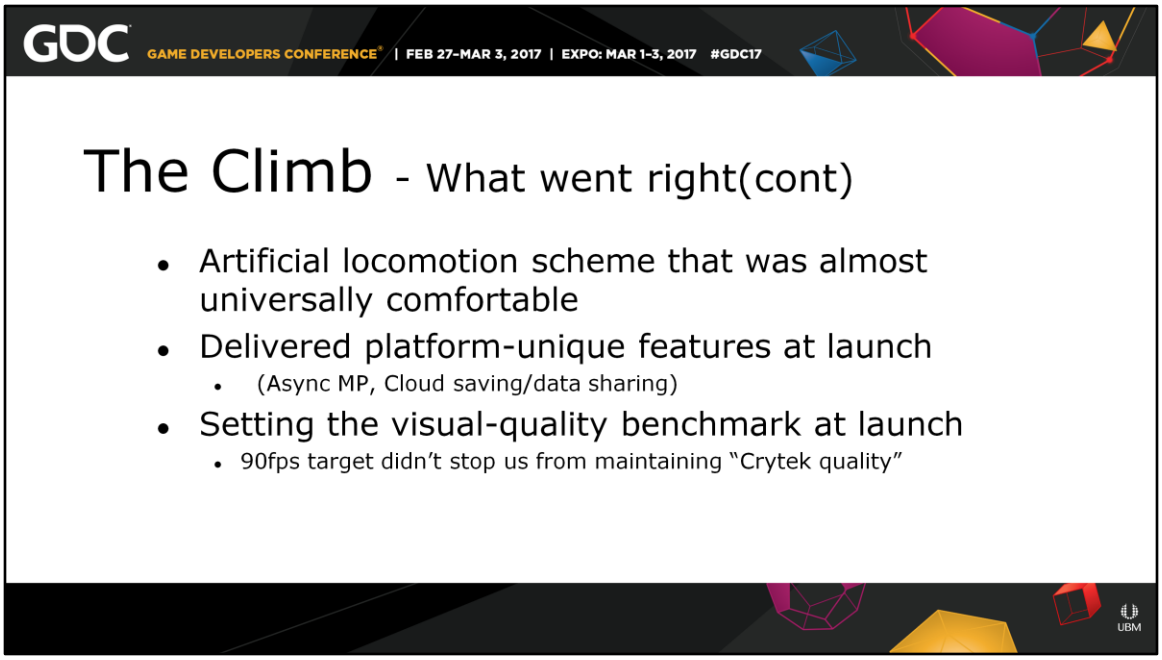
The strict delivery plan coupled with a strong vision – left less room for scope inflation or feature creep, helping to keep project on track (and within bounds).

Hitting performance targets early, and well – meant we had a game that ran close to 90hz on target spec for most of the project. Optimization cycles were relatively painless as result, and mostly driven by the content teams, with less intervention by tech teams needed.

Having a milestone on almost monthly basis, (and mid-deliverables every 2 weeks) also led to high stability – the game rarely crashed, and when it did, it was mostly trivial changes to fix.

Pace Of delivery

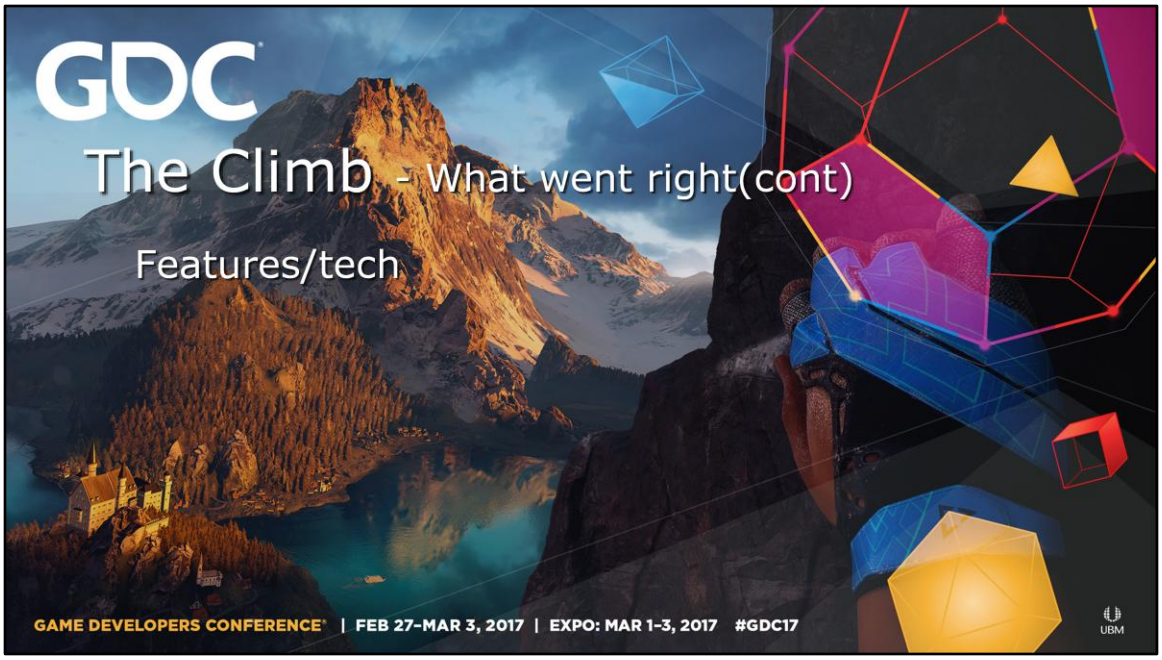
Development was smooth enough to allow additional prototype development – Climb being a launch title would require to ship with Xbox controller as primary input device, but having the core-game up and running for majority of the development allowed for additional time to be spent on the yet-to-be announced Touch controller implementation (the game was already fully playable with touch in spring 2016).



The main star of the game was the locomotion scheme, defying the established logic of artificial locomotion in VR not being possible in near universally comfortable way.

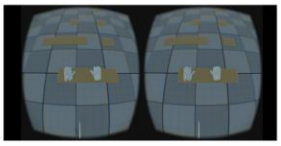
Planning for the entire development early allowed us to side-step external dependency problems. For example - Initial plan called for Async-multiplayer, and with Oculus online-platform still in development, we weren't sure what features it would provide for the launch window. Decision was made to offload most of the required feature-set to our in-house backend team to avoid last minute external dependencies there. This would turn out to be a good decision, and we were the first Oculus Home title to launch with cloud-saves, in addition to an elaborate achievement system and leader-boards supported by our in-house online-backend. If we hadn't planned ahead, we probably wouldn't have been able to check off so many features on the list.

Even though we worked on a brand new medium with unfinished hw and sw platforms for most of development, many firsts were delivered along the way. Being a CE title, there was no historical precedent for launching at 90fps on target spec before – and team delivered it without any real compromises to the quality of the vision. Performance targets were made additionally difficult by evolution of platform APIs and SW which continually shifted ground as launch approached (one day ATW and Queue Ahead would improve things, the next we'd be running into VRam trashing because Home application was consuming too many resources), but the strict approach to monitoring performance helped.


GDC
The Climb - What went right(cont)
Features/tech
GAME DEVELOPERS CONFERENCE | FEB 27–MAR 3, 2017 | EXPO: MAR 1-3, 2017 #GDC17
UBM

The Climb - What went right(cont)

- Locomotion in Climb
  - Successful prototype established the core ruleset
    - Based on "real-world" behavior
    - "Impulse" motion, short-cuts on rotation/animated motions
    - Rotations inwards best avoided



Locomotion in Climb has a simple origin – during early prototyping stages, one of our designers was trying out what it would be like to do actual climbing in VR. An avid climber himself, he worked backwards from the actions performed in real life – look for hold -> stretch/grab -> pull yourself towards the new hold, and implemented a simple prototype of that very mechanic in flow-graph. The mechanic not only worked – but it was almost instantly clear to new users that tried it : look – press one button per-hand to grab and pull, release when you're ready to look for next hold.

Analysing from successful prototype backwards, we derived a few elements that help with comfort aspects of this mechanic:

-By looking at the destination before initiating a move, we ensure that movement direction matches that of user's focus. Moreover - in many situations the object user is focusing on (the handhold) will be kept mostly in the center of their vision for duration of the said move, avoiding the issues you'd normally get by panning next to a wall (this also proved important for rotations).

-Movement is an "impulse" – a curve with a short burst of quick acceleration/deceleration at its edges.

-Movements are kept relatively short

-When rotation is involved due-to-handhold orientation, rotational movements are performed in even shorter bursts – minimizing the time keeps brain from "sensing" discomfort that would normally occur during the same rotation if it were slower. Note that rotation and translation curve need not be synchronized here (it's perfectly fine to complete rotation faster while the rest of motion continues on for longer)

-Rotations inwards are best avoided, or kept at very minimal angles (this works fine for most climbing situations, and is easily mitigated in level design)

The Climb - What went right(cont)

- Locomotion in Climb
  - Climb final move-set is basically platforming
    - Jumping, "riding pulleys", ledge-pulls, falling…
  - Motion controls were "easier"
    - Changes how player/world intersections work

Putting it all-together, we ended up with something very much like a platformer move-set, and later uses in Robinson and DLC releases for Climb would take more and more liberties in exploring the extent of possibilities the system afforded, as level designers got more and more comfortable with it.

Moving to motion-controls, a lot of things were "easy" – we keep the grip mechanic forcing mild-rotational locking on the player, but put all movement into player 1:1 motion control. Jumps work both physical (by throwing yourself) and via button press, and the other mechanics remain largely unchanged.

The Climb - What went right(cont)

- Rendering long vistas with dynamic shadows
  - Near vs far detail
  - Fully modeled 50km vistas with lots of verticality
  - Light & shadows stay dynamic
    - 8 cascades, start@3m, logarithmic expansion, 1024x1024 each
    - No cheating with "shadows vanish after N-th cascade"
    - Shadow-cache to combat CPU bottlenecks
    - Cascade-shadow stability in VR

In VR, player can always stick their head right into objects, necessitating near-plane to be moved to suitably tiny values(cm range) – be mindful of depth precision impact here, and near-field objects to be rendered appropriately detailed. Climb of course, also highlights the long-view distances at the same time.

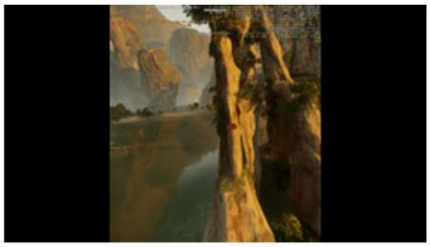
While vistas in Climb are temptingly like sky-box at first-glance, movement verticality highlights massive amounts of parallax in the surrounding scene, so actual geometry was preferred for the entire level. This in turn meant we had to light and shadow them in runtime – as in CE we don't actually do any sort of baked solutions to illumination, and it's not really in Crytek DNA to accept shortcuts for this. If that wasn't enough – as the old saying goes "give the art-team a finger and they'll take off your hand" – the lighting design for the 15 levels shipped in release version of The Climb actually called for dramatic, near-horizon sun location at least once per setting, generating a nice worst-case scenario for runtime shadow-maps.

At 50km, we've used up to 8 cascades to cover the terrain (a LOG2 setup with "fair"-precision shadows in near-cascade of about 3m, 1024x1024).

Rasterization of such distances turned out to be a fairly CPU limited affair, which is where cached shadow-cascades were used to improve performance.

The Climb - What went right(cont)

- Runtime occlusion in VR
  - Rely on LOD instead

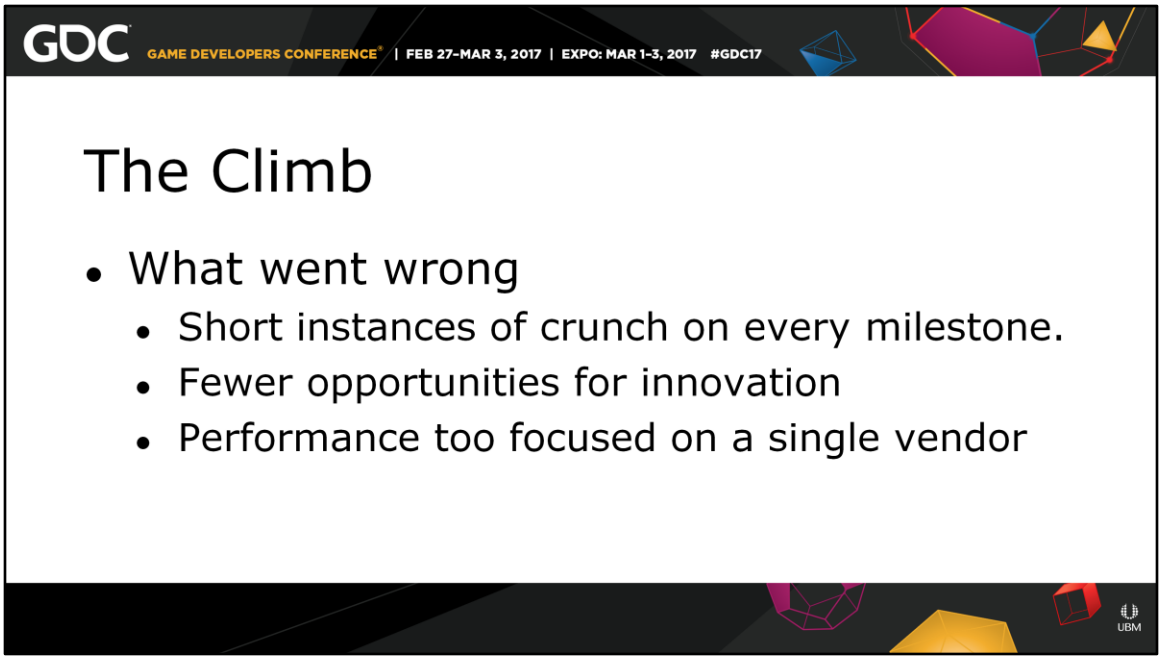


Runtime occlusion translated poorly to VR due to temporal dependencies in CE implementation. Game design worked in our favor here however, and mainly moving player along the walls, we were able to use very aggressive LOD as seen in this clip, without player ever being the wiser, and still presenting a stable VR world.

<Details>

Runtime occlusion was another aspect that did not translate from non-VR -> VR at all. In CE, the main driver of occlusion is queries based on reprojected depth from previous frames, which in VR yielded somewhere between “no-improvement” to “visible pop-in”, with little in between. Factoring in the fact that Climb level design puts player in front of a wall on one side, and a completely open landscape on the other, the benefits would have been low anyway – so the choice was made to disable runtime-occlusion completely, and tackle worst-case scenario (if any occurred) manually from level-design side. Predictably – the aforementioned level-structure meant there was little room for manual visibility setup in Climb either, but we would end up using the approach a fair amount in Robinson, together with some degree of runtime occlusion.

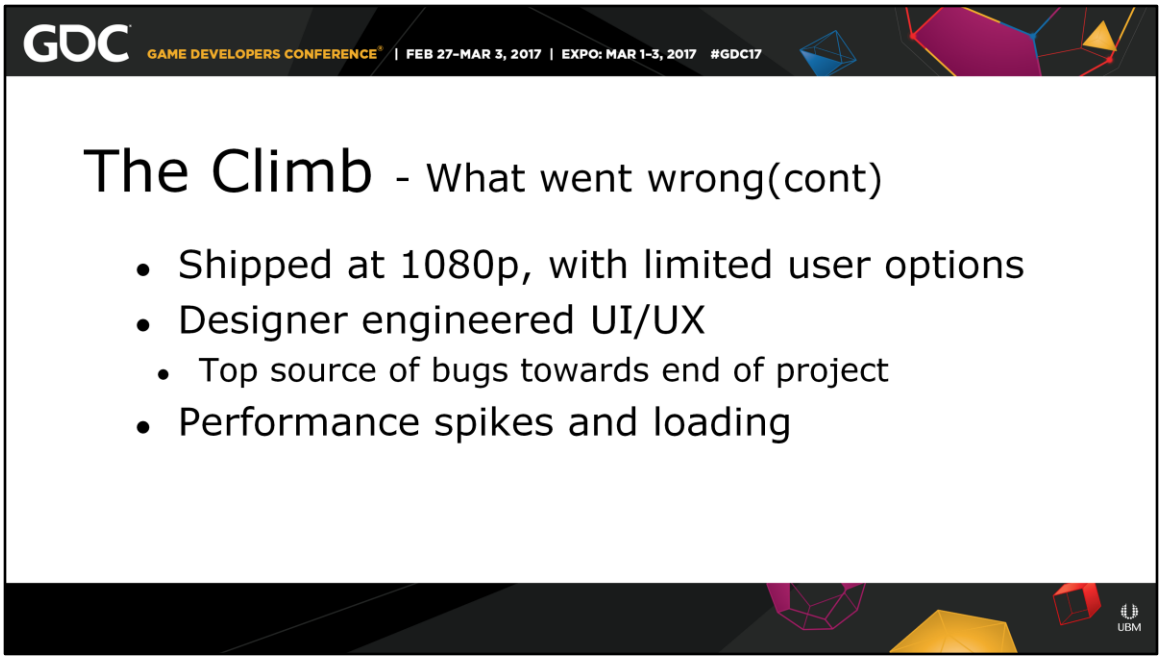
However – that same level structure ultimately turned out to be key to maintaining high fidelity close-up. Having player face the wall most of the time also meant much less to worry about for hiding LOD transitions of traversable terrain, and thus much less need for occlusion in the first place. Outside of few instance in later levels, Climb maintains an illusion of a very “stable” world LOD-wise, an illusion that quickly shatters if you simply pan camera backwards in free-flight mode.



While delivery pace was rapid – we ended up with small crunch-periods on almost every milestone – this was in part caused by team overcommitting and very high-internal quality standards, but nonetheless, the effects of having it periodically occur over 10 months started to show towards the end of the project.

Opportunities for scope inflation were controlled by detailed plan – but on the flip side there was less room for risking any innovation also. We mostly took a conservative approach to any tech changes, avoiding new-tech-dependencies in favor of stability & predictability of the toolset that the team worked with. Not being on the absolute latest engine-code did limit the support from core-engine team for troubleshooting somewhat, but the codebase was also stable enough for Climb team to mostly tackle any issues on its own.

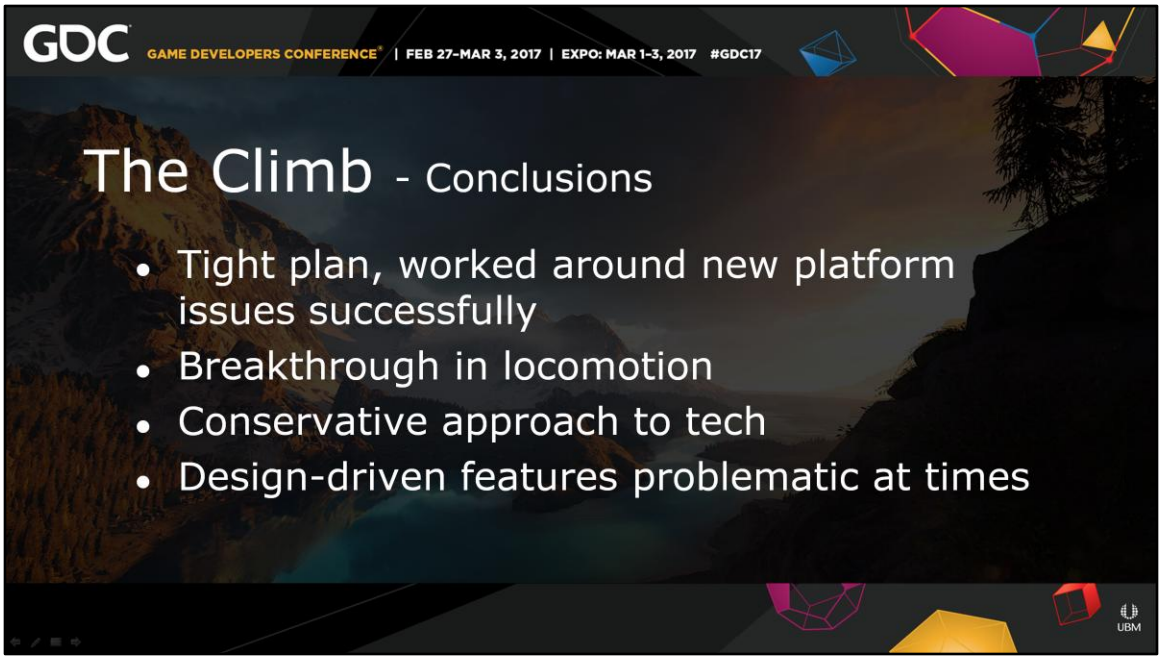
While optimizing performance, we focused on a small selection of GPUs and expected that the “equivalent” hardware from other vendors would behave similarly. But late testing had us running into problems and until a point very near release it looked like we weren’t going to hit the target performance there. Lessons learned about assumed equivalencies.

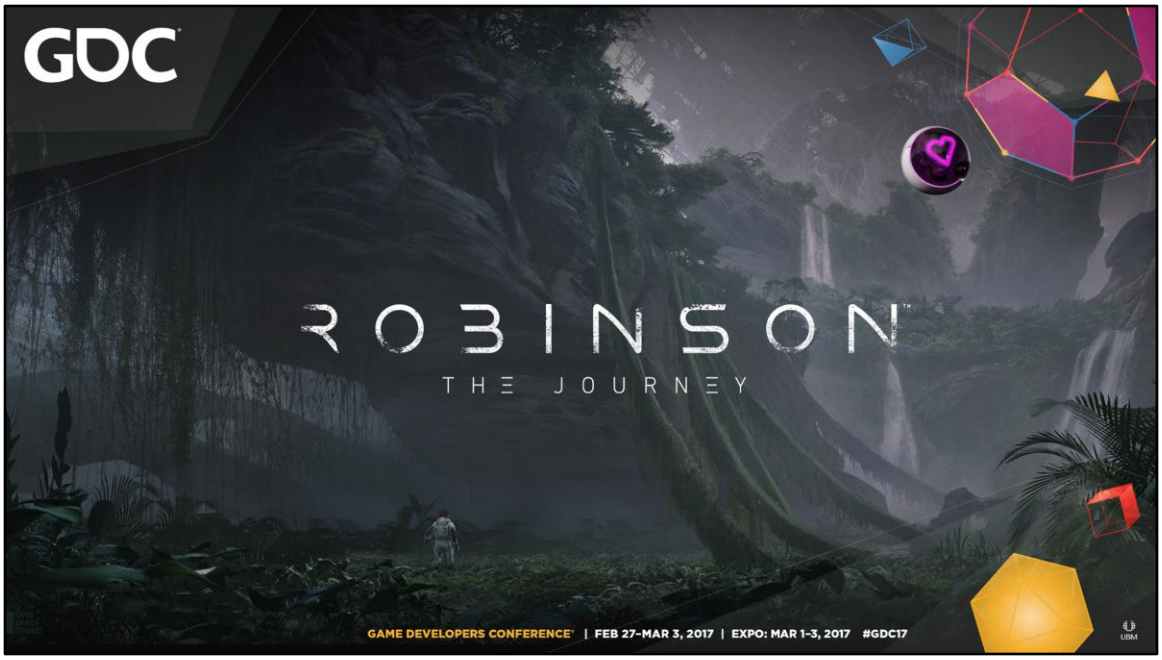


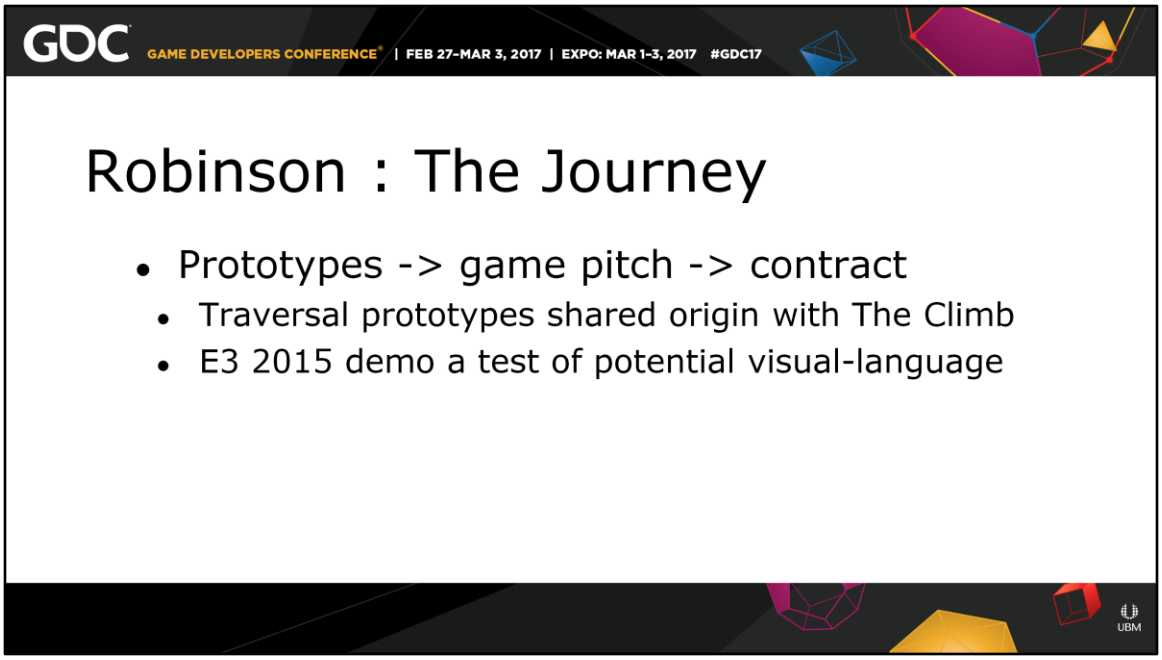
Maintaining strict control over performance also meant that in light of hitting shipping dates, we chose to restrict user-options, and ship with only two configurations (Default, and "low spec"). This included restricting resolution to 1080p, as simply too much testing went into that to risk other configurations before launch. A patch would eventually get released to give proper user-control, but only months after the fact.

For UX, we've opted for mostly designer controlled workflow with minimal code dependencies, in no small part also because VR UX, as so many other things, was completely uncharted territory at this point, and we wanted to leave options open for experimentation as much as possible. This worked well on initial scale prototypes, and yielded some really good looking and functioning behaviors, but as the menu-system grew (one of the few areas of The Climb where scope did considerably outgrow the initial plan), it quickly became one of the top bug-contributors in the project.

While overall performance targets were met mostly with ease, running at 90hz exposes many subsystems that look harmless at 30fps. Climb didn't encounter significant problems here – but "loading into maps" which is actually just a streaming scenario where world is already fully playable, would introduce noticeable spikes along the way, which forced us to extend loading-screens in front of starting a map until framerate stabilized enough for player to enter.

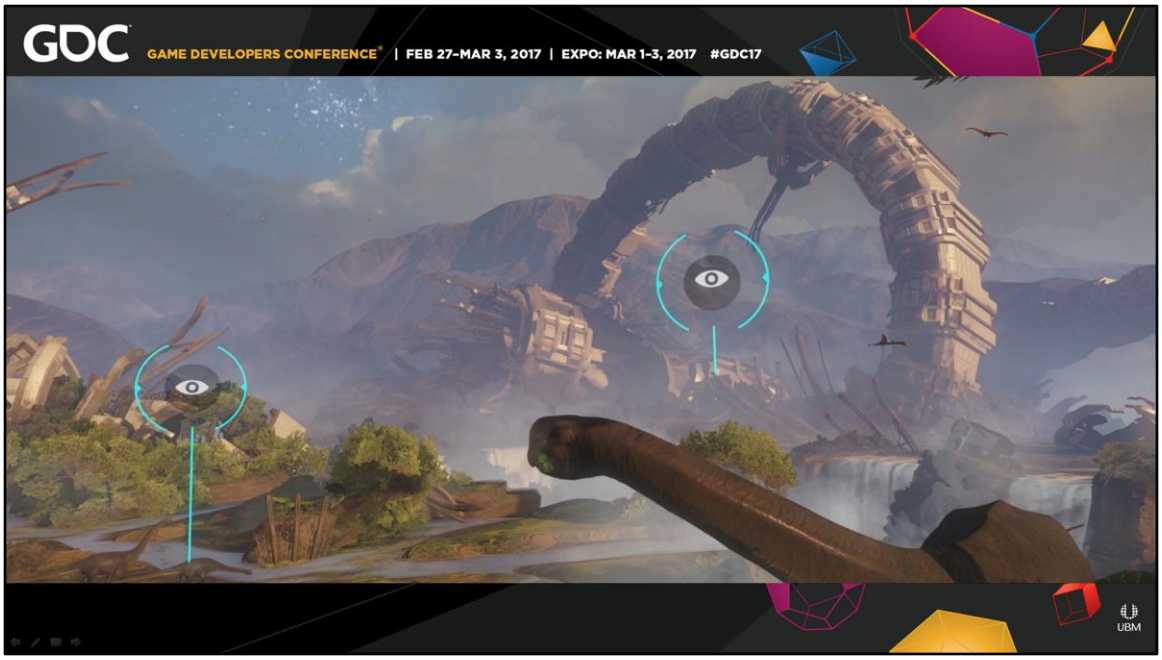

GDC GAME DEVELOPERS CONFERENCE | FEB 27-MAR 3, 2017 | EXPO: MAR 1-3, 2017 #GDC17
The Climb - Conclusions
• Tight plan, worked around new platform issues successfully
• Breakthrough in locomotion
• Conservative approach to tech
• Design-driven features problematic at times
UBM


GDC
ROBINSON
THE JOURNEY
GAME DEVELOPERS CONFERENCE | FEB 27–MAR 3, 2017 | EXPO: MAR 1–3, 2017 #GDC17
UBM

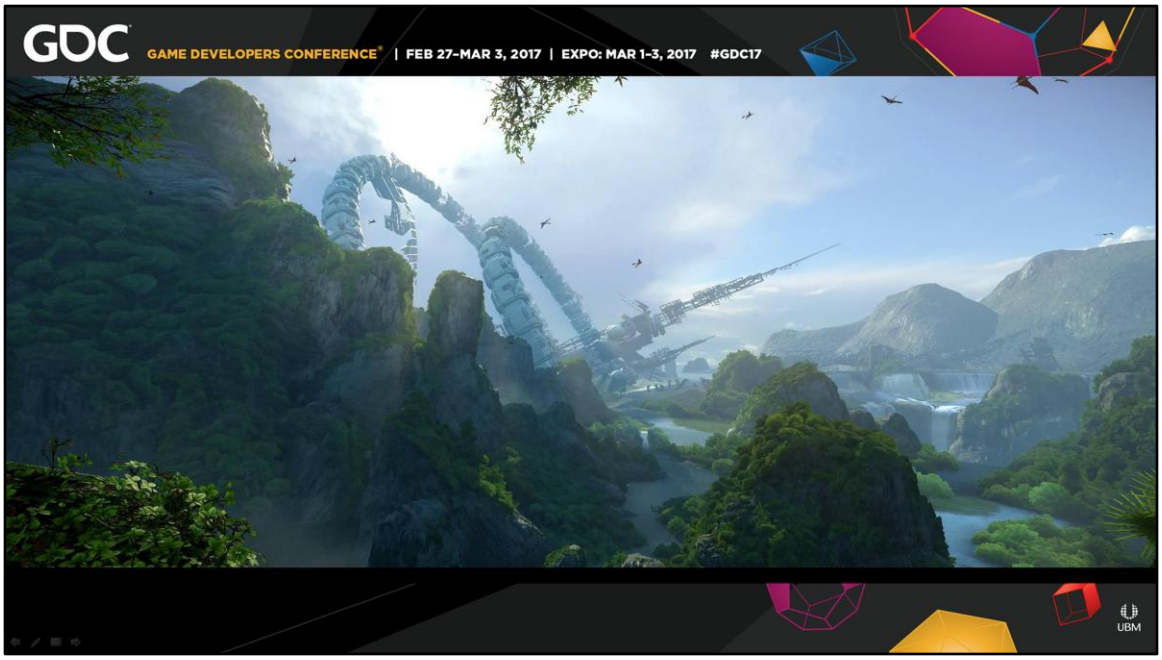


In terms of public announcements – Robinson project started life before The Climb, being announced at the very E3 that marked the start of development of prior title. This however, also marks the stark contrast in how the IP was conceived, and eventually turned into a game title. Early locomotion experiments were in fact tied closely to the idea of bringing a larger-scale, open world exploration project to VR, and around the same time, origins of the sci-fi story that makes up for the background of Robinson : The Journey were being thought of.

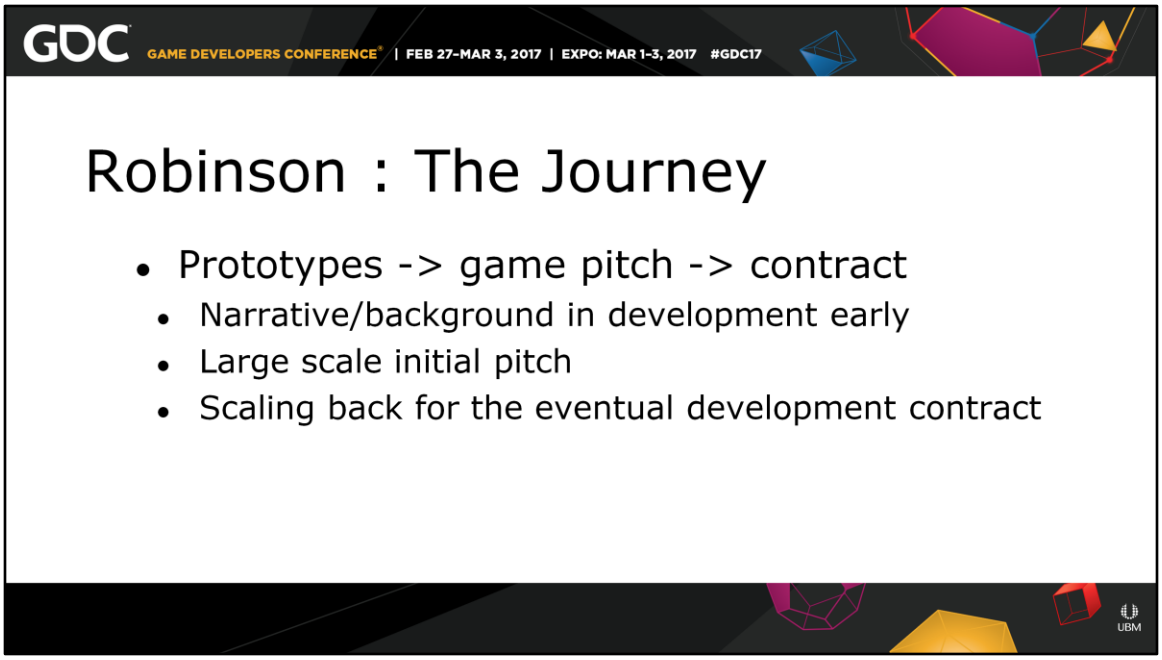
Much as E3 2015 demo was a first time presenting an artificial-locomotion solution in CE, it was also first visual style test / what-if of a potential world players could visit in Robinson. And the slew of early artificial locomotion prototypes that gave birth to The Climb mechanics was also the first draft of exploring possibilities of brining Robinson to life.



The early visual-language from E3 2015



And the final visual designs at release in Nov 2016



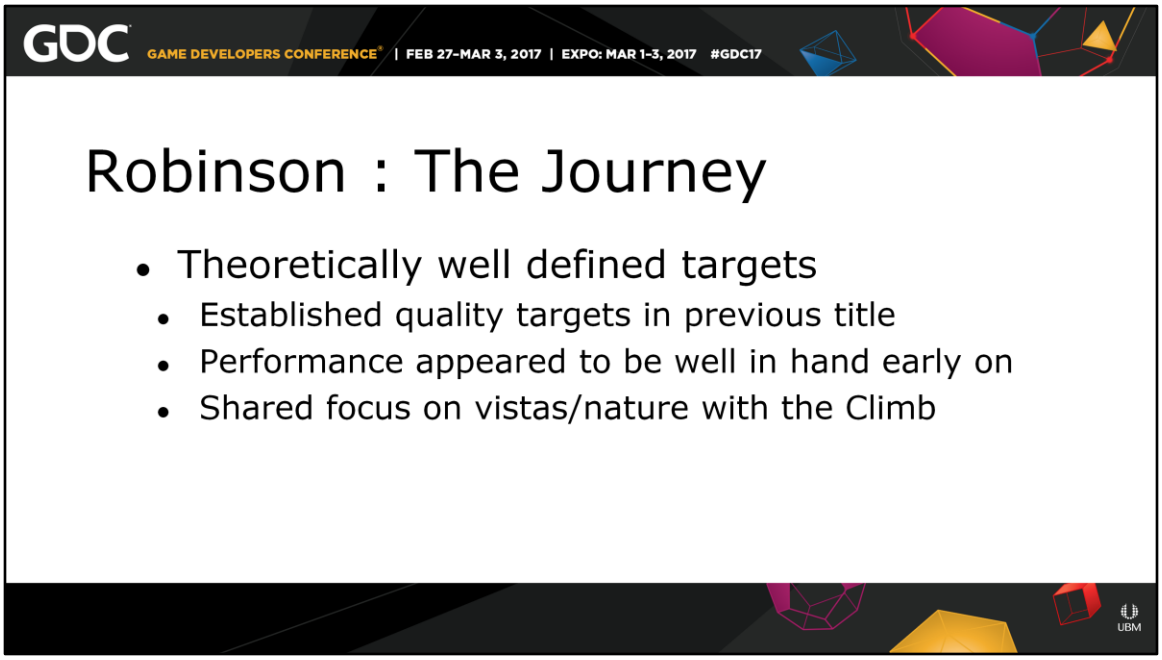
The initial pitch for the game would mark the difference between two games even more clearly – Robinson was envisioned very much along the lines of “traditional” retail/AAA with 2 year development cycle and proportionate amount of content (and development cost). This would go through many iterations – ultimately the game agreed upon with the publisher would be set at 30-60minutes (with options for future expansions).

Robinson : The Journey

- More aggressive timeline, but more freedom on deliverables
  - Initial pitch called for a 6month dev-cycle
  - Less detailed milestones, and fewer of them

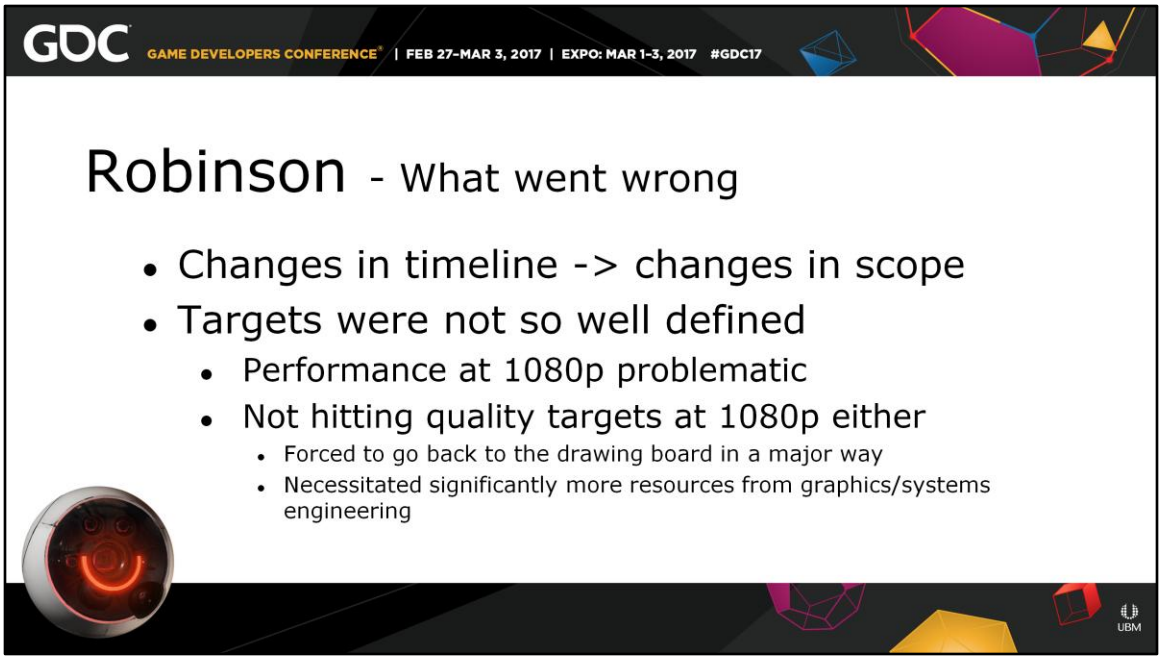
Initial timeline would reflect this smaller scope – expecting a deliverable as early as mid June 2016 (around 6 months). Note that this was before release dates were known for the target platform.

Milestone plan for the project was however, considerably more loose and less detailed than Climb's, leaving more freedom to the team to define deliverables as we went. However, while having less constraints looked good at first glance, the limited timeline didn't exactly leave a lot of room for experimentation.



One initial perceived “advantage” was that Robinson team already saw most of the lessons Climb had gone through by the time production officially started, and performance/visual implications of VR were well understood. 1080p was chosen as a “good” resolution target (with expectations that up to 30% lower could still be “acceptable”, based on experience with Oculus devkits) and PSVR allowed us to target 60fps with reprojection, reducing performance pressure on PS4 hw.

Level design called for some changes with having actual on-ground traversal, but there was still the intent on running with mostly nature-settings and relying on long-view distances and vistas to emphasize the scale and wow-factor of the world.



We'll start Robinson with What went wrong – due to the contrasting nature of development with respect to the Climb.

When launch date was decided for PSVR – timeline was extended to October. This would in theory be a good thing for the development schedule – but somewhere along the lines, scope started growing as well. By the time we shipped, the game ended up at 4-5hour mark – nearly 5x longer than initial plans.

Performance and resolution targets turned out to be significantly off-base in places – in-short, our "prior" experience that 1080p looks sufficiently good on one VR device did not translate to PSVR at all*. This was further complicated by the fact that performance at 1080p was proving difficult as well, leaving no simple solutions of balancing of the two. We were forced to reinvent, and the graphics/systems engineering needs on the project escalated dramatically as result.

*While 1080p still looks nice on Oculus CV1 and even 900p was acceptable in some demos – PSVR display characteristics are much less friendly to missing pixels(the main culprit here is tighter packing of pixels on PSVR – more noticeable "screendoor" effect on CV1 tends to "diffuse" the display image, reducing the impact of lower resolution). 1080p would be the resolution that Robinson was first publicly shown with at E3 2016, and while general impressions were favorable – the over-blurring of the image turned out to be a frequent complaint. This also meant that lowering the resolution any-further was completely off the table, throwing most of our early performance targets into question.

All this forced us to re-evaluate a lot of the expectations and existing technology in use, at a significant cost/effort investment. Systems/graphics engineering support needs for the project grew dramatically as result.

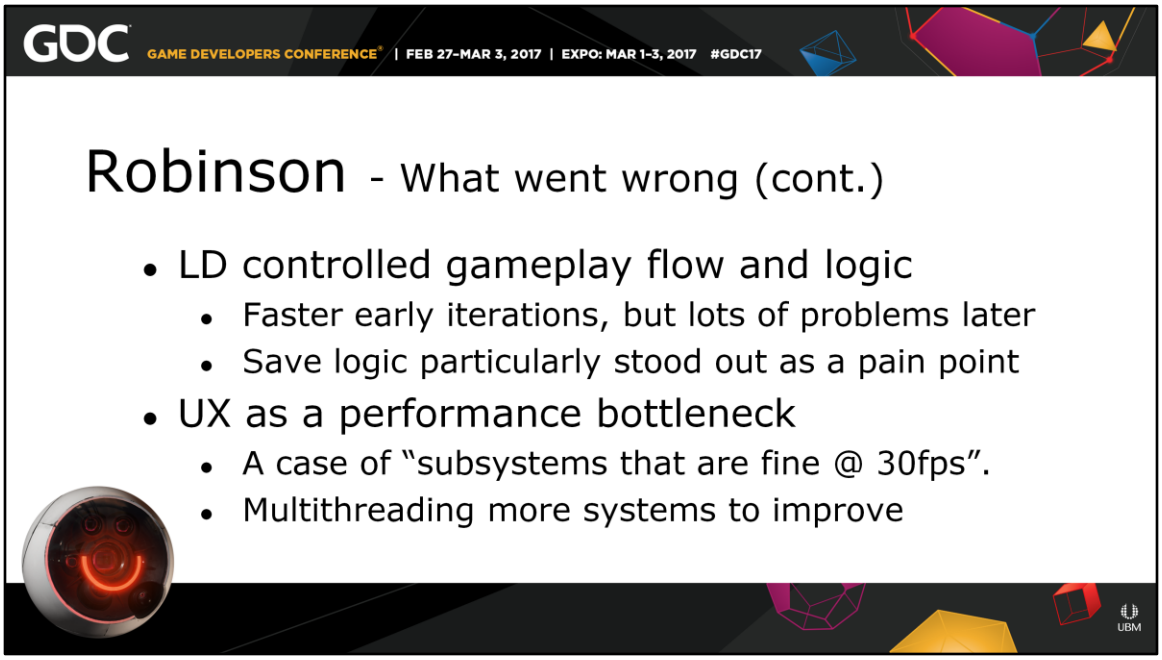
Robinson - What went wrong (cont.)

- No well defined core-mechanics at first
  - We had various prototypes and concepts, but no core-loop
  - Scope shrinking also left a lot of the initial gameplay goals on the cutting floor
  - Heavy reliance on AI
    - Significant team-growth on AI programming in particular

Gameplay didn't fare much better early on – unlike climb, early concept work did not yield a viable core-loop (and admittedly – with initial relatively small scope, this was viewed as less important).

30-60minutes inside a sandbox of prototypes was a viable option – but that changed when timeline (and scope) grew.

AI (especially both of the player companions) played a major part in early designs, but that again, was underestimated when extending the functionality to full game. AI programming team would be one of the largest groups by the end of the project.



Note to self – Call Level Design by name not abbreviation.

Again – under original assumption of a shorter experience, direction was chosen for Level Design to control most of the gameplay flow and associated logic (including things like save mechanics etc.). This was also in line with perceived limitations of available gameplay engineering resources. While this gave us a significant boost in terms of iteration speeds early on, having non-engineers craft larger systems quickly became problematic as the game (and scope) grew.

Save logic in particular would be one of the most painful sources of bugs towards the end of project, resulting in many progression blocking issues that had to be fixed in final weeks.

PS4 specific code path was missing async-saving support when Robinson development started. The major problem however was Level Design controlled progress saving, which didn't leave much control over saving process in the code.

Along similar lines as in The Climb, UX was heavily design driven as well. We had allocated more engineering support this time, and things learned in The Climb helped avoid many of the issues, but reliance on Flash turned out to be a significant source of performance problems towards the end of the project. This was another case of a subsystem that generally doesn't cause issues at 30 but becomes visible at higher framerates.

Obviously shows more on platforms with fewer CPU resources.

Robinson - What went wrong (cont.)

- Building the plane while flying
  - Some of performance tools failing mid-development due to new changes
  - Additional costs of maintaining workflow stability

Robinson relied on far more “in-development” technology than before (opting for using latest CE 5.x rendering features, as well as a host of new improvements developed specifically for it). This was necessary to achieve results we were targeting – but it led to additional stability problems with the codebase that became particularly prevalent in second half of development. Built-in profiling tools were an example of this – as changes to stereo-rendering code-path broke things leading to misleading numbers multiple times in development.

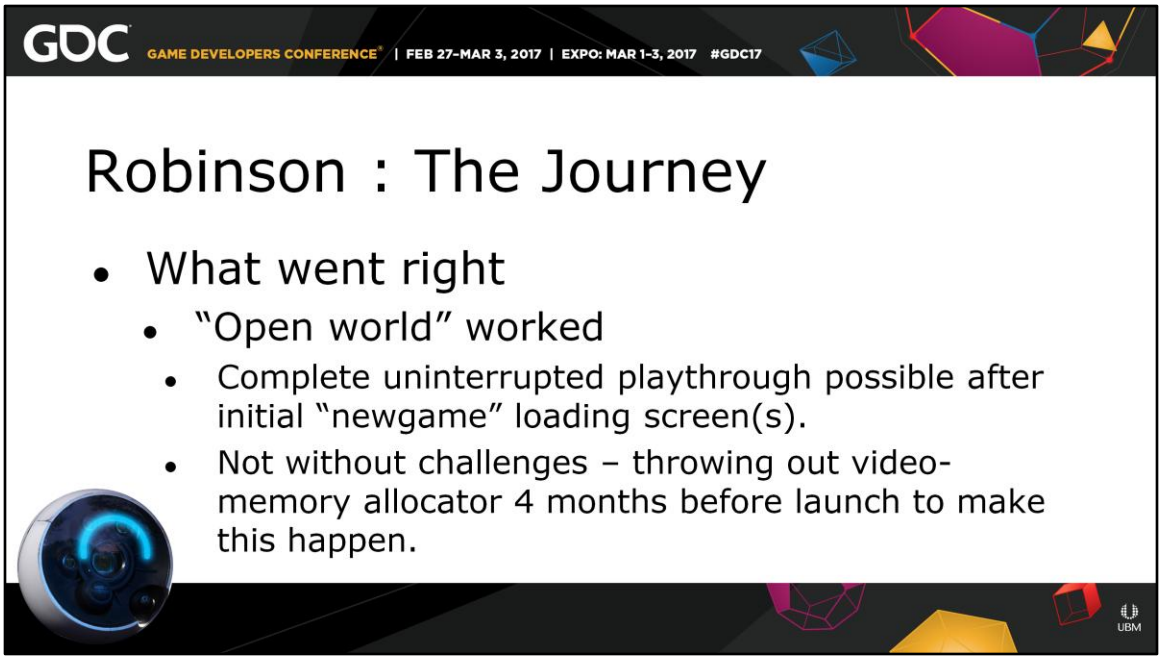
Robinson - What went wrong (cont.)

- Hitting performance targets late
  - Switched to new (CE5.x) renderer after E3
  - Many performance targeted features came late
  - The first time entire game ran at (mostly stable)60fps was at Beta
- Locomotion options

Robinson's first public outing (E3 2015) started some 2 months before E3, when the level barely ran at 15fps. By the time E3 came around, we were somehow scraping by at 50-60fps, with some of the most aggressive LOD used, and running at only 1080p, lacking in image quality as well. Past E3, the project would switch to the new CE 5.0 renderer, which along with many other improvements coming in the next few months, would finally allow us to hit all the targets.

First build to reach mostly stable 60fps (less than 10TRC frame-drop notifications over entire playthrough) was in mid-September, right at our Beta milestone.

In interest of giving users more freedom to choose – Robinson shipped with options to disable most "comfort" restrictions. Unfortunately they were never explicitly marked as such – leading to a portion of the userbase attempting to use locomotion that is specifically not likely to be comfortable for majority of player base.



Climb was at its heart – a level based game, not unlike a racing experience – large, basically open-traversal levels, but nonetheless.

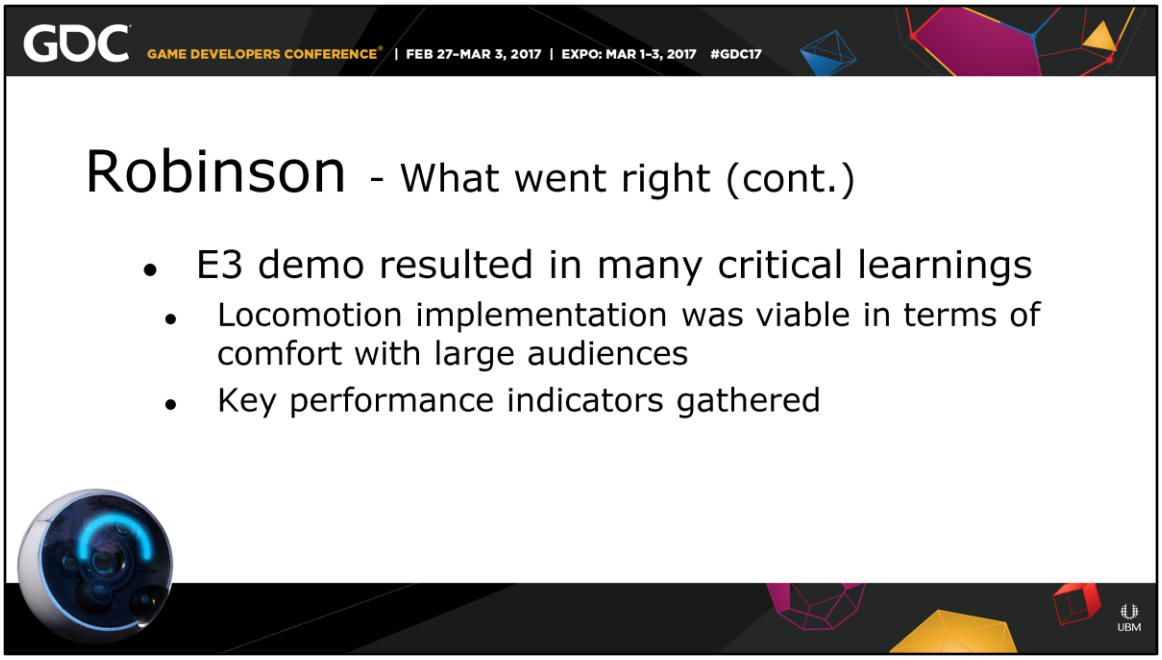
Robinson started out in levels – but the ambition was always to present a continuous world to the player to explore at their leisure. Through a combination of new memory allocator improvements and a few other changes that ironed out I/O spikes we were able to achieve that – final game loads once when starting a new game, and from then on it's possible to complete the game without encountering a single load screen. As in climb – any subsequent loading screen (death, fast-travel) are really just hiding texture/object streaming, the game is always running and playable in the background.

Robinson - What went right (cont.)

- Well structured art workflow
  - OBM/POM worked well for micro-scale detail in VR
- Comprehensive/strict performance targets for content creation
  - Less issues with shadows and lighting

Art workflow in Robinson was thought through well in advance with respect to platform we were targeting. This involved things like favoring texture detail over geometry and object complexity (due to limitations of earlier tech – this was less important in final renderer), photogrammetric workflow, utilizing Occlusion Bump Mapping for ground detail and Parallax Occlusion Mapping in select areas where that wasn't enough, etc. There was very little friction between this workflow and the constantly in-flux tech, which was a testament to it being a smart choice.

Robinson also had very detailed metrics to work with (in part also thanks to being on closed platform). Art teams were given strict – per-ms budgets to work with in the final months, and combined new draw-call budgets in CE5.x – we were able to actually push art quality considerably further than early results indicated. Unlike the climb – we've also experienced considerably less problems with lighting work and shadows. The former benefited from new renderer improvements, and the latter was greatly simplified now that we knew exact solutions to each of the issues encountered before.



E3 demo came through again. Like with 2015, E3 2016 demo gave us many valuable lessons for remainder of development - both about what worked, and what didn't – yet. Locomotion in Robinson was one of these examples, as until E3, the choice of free-roaming on-ground locomotion was still very much under scrutiny in regards to comfort issues.

Performance indicators gathered for content creation and other aspects were crucial to set-ourselves up right for the final stretch of development – all the content built from this point onwards was measured against them.

Robinson - What went right (cont.)

- Profiling performance and memory
  - Platform tools worked well
  - Art teams adopted "programmer" visual profiler successfully
  - Track memory on-screen for OOM crashes
  - New feature to profile spikes saved the day

We could spend 10 slides just talking about everything that went on (and right) with performance profiling in Robinson – but really the following were key.

Platform tools were very helpful to keep ALL teams in-line – TRC notifications are a great way to do this (would love to have the option to "enable" this for normal games as well in the future – Sony? ;)).

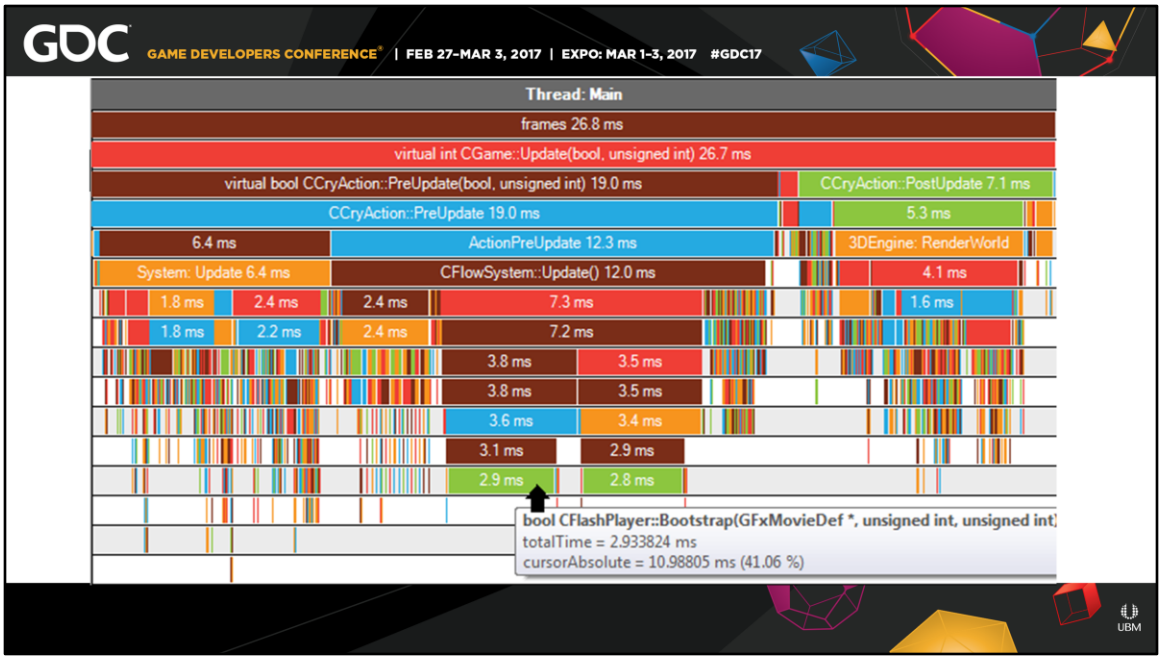
Content/art teams successfully adopted built-in visual engine profilers that were mostly aimed at programmers to date, measuring their budgets down to milisecond, and basically making our jobs a whole lot simpler.

Out of memory crashes happened – but tracking them with on-screen memory and info "screenshot on crash" made chasing them down a lot simpler.

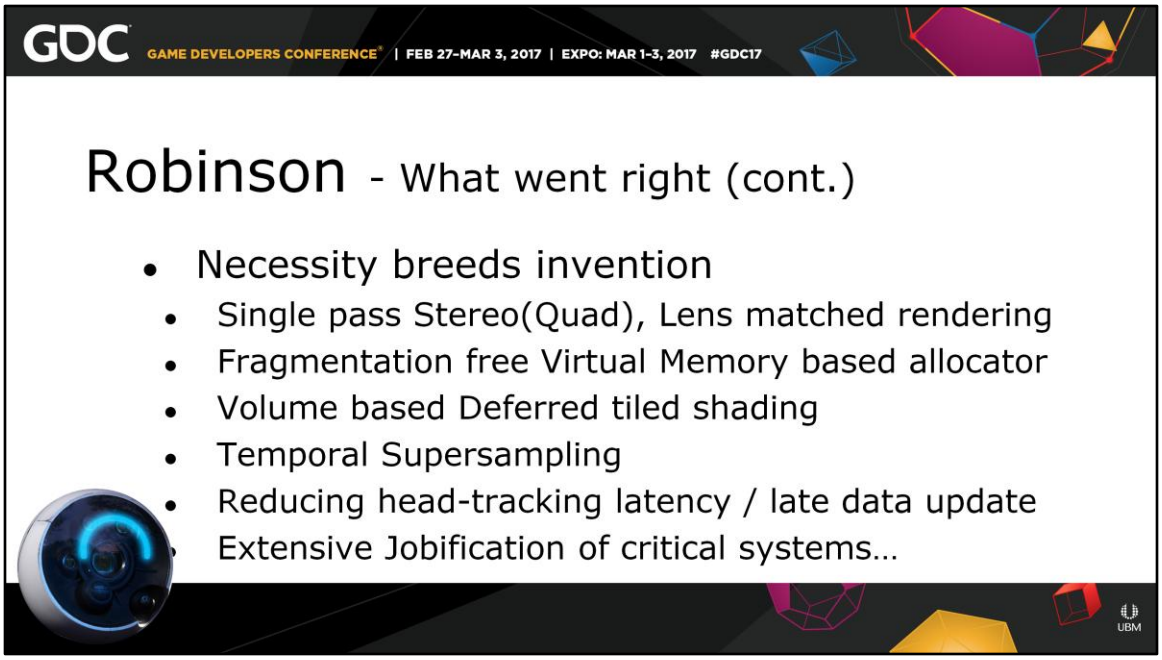
- In between alpha and beta - OOM crashes

- In release build almost 0.5Gb of free mem

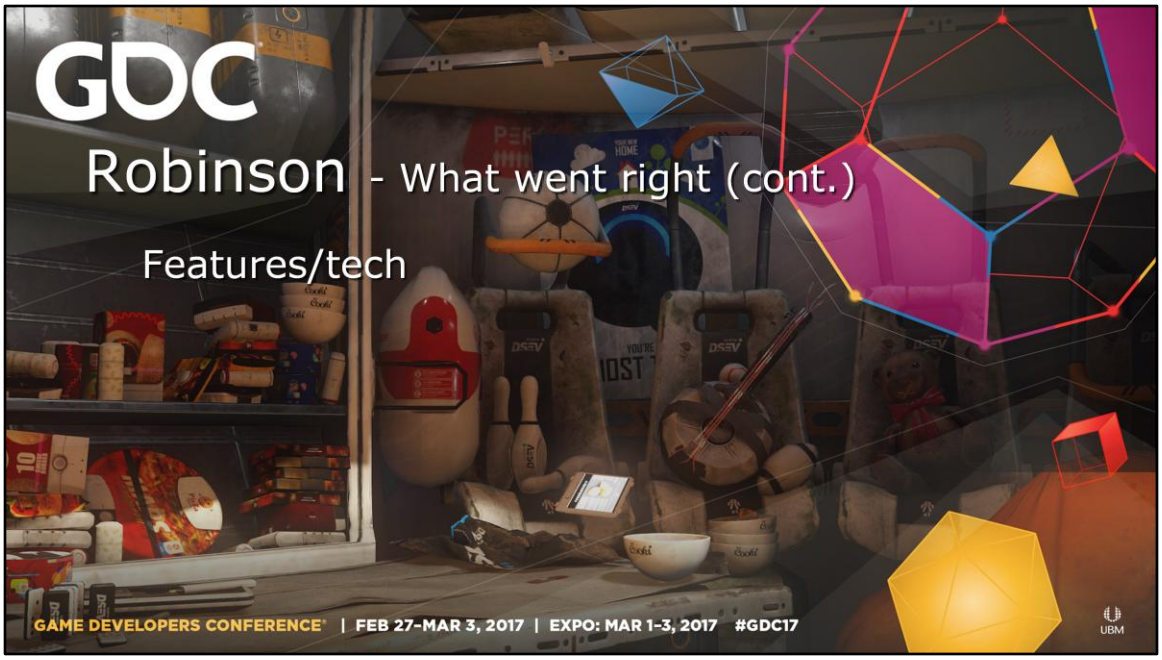
But perhaps most importantly – extending built-in profiler to dump data on "hit-X ms budget" was the biggest win, making short work of issues ranging from streaming I/O, flash processing, level script processing spikes, and many others – all systems that would perform "within budget" on average, but exhibit irregular spiky behavior due to lazy evaluations and other elements.

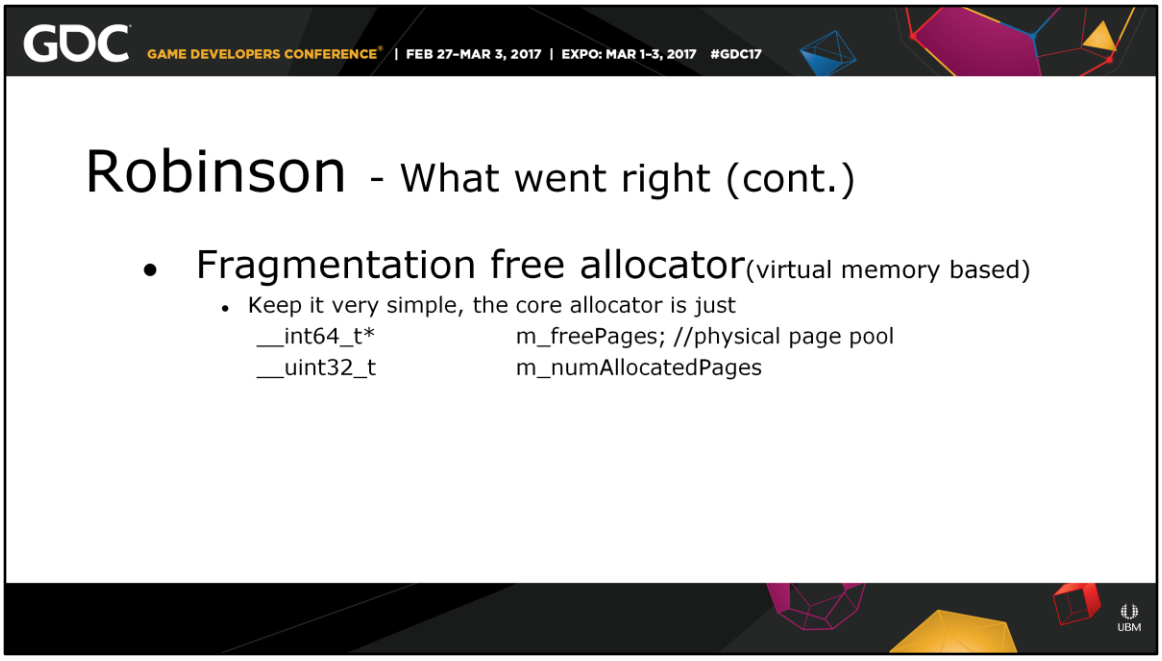


A snapshot of a profile dump on spike.



Being faced with multiple issues, from expanded scope to not so accurate original performance estimates, we were forced to reinvent. The list of things here is only a sample of the larger changes introduced during Robinson, many of which came through in the final 4 months of development. We'll dive deeper into a number of these in follow-up slides.


GDC
Robinson - What went right (cont.)
Features/tech
GAME DEVELOPERS CONFERENCE® | FEB 27–MAR 3, 2017 | EXPO: MAR 1-3, 2017 #GDC17
UBM



Running out of memory caught us by surprise on Robinson – mainly because we were coming off a title that already fit fairly well in the predicted budgets of PS4 ram, and more so because the initial causes of memory problems were the one subsystem that is not supposed to fail (texture streaming cache). The problem turned out to be a combination of underlying assumptions in the said system (PC centric architecture, fragmentation largely being a non-issue on PC due to relaxed memory constraints, and defragmentation policy only implemented for PC platform anyway) and a few bugs with the heuristics (eg. the cache could get "stuck" on texture mipmap-unloads as it requires a new-allocation before it can free-up any portion of mip-chain).

Regardless of the cause – we had to adapt to the situation – and one of the plans put into action was taking another look at how we handled **graphics-allocations** on PS4.

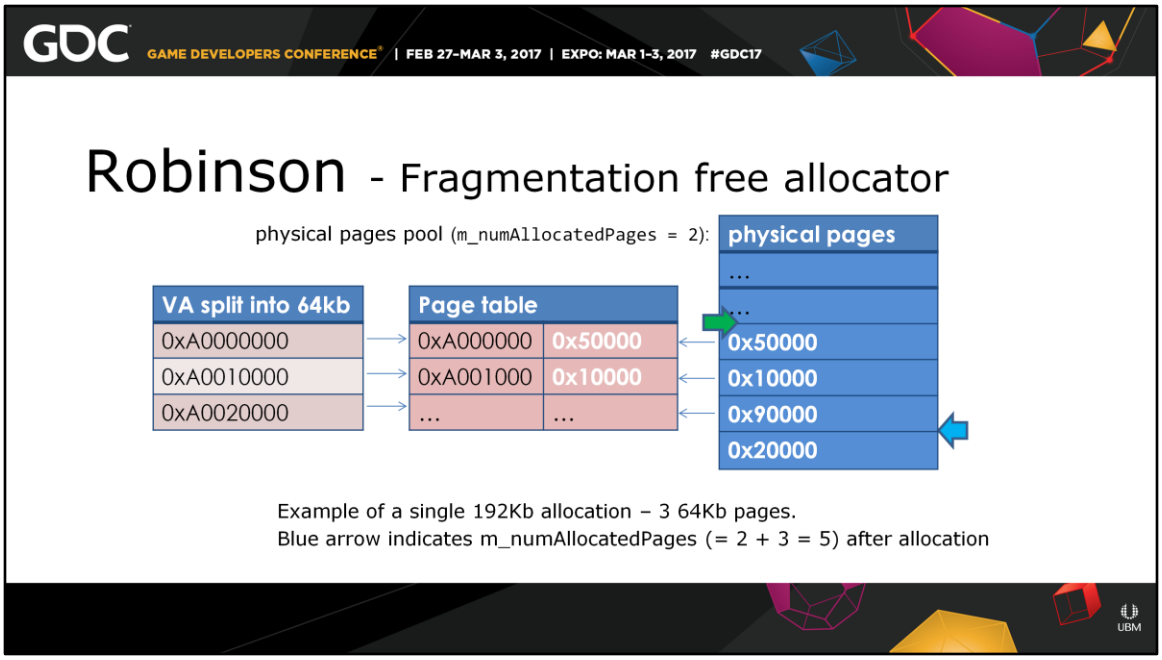
<Why Virtual memory based allocator?>

PS4 is one of the first times we have the entire hardware capable of addressing entire virtual-address range, and comes with explicit APIs to control this. Available address range is in excess of 400GB – which basically gives us space to fit several games worth into it – and then some.

a) Concept is actually remarkably simple – the core allocator just keeps an array of physical pages and a count of allocated ones

b) We decided to allocate a fixed size pool of physical space for video memory, so we have full control of memory usage (in more general case the pool can be dynamic). Note that at this point – none of the allocated physical memory has been mapped into virtual-address range.

c) The pool is split into 64KB pages, as noted above, and initial state of pages is in-linear order, though that is unimportant in use



A quick example of allocator in action. Notice that pages are shuffled in pool. They are no longer linear – this represents that allocator was in use for some time.

Virtual address is always continuous. On allocation we simply map continuous virtual space to availiable physical pages, which can come in any order.

Deallocation happens in a similar way: physical pages return to the pool, `m_numAllocatedPages moves to the left.`

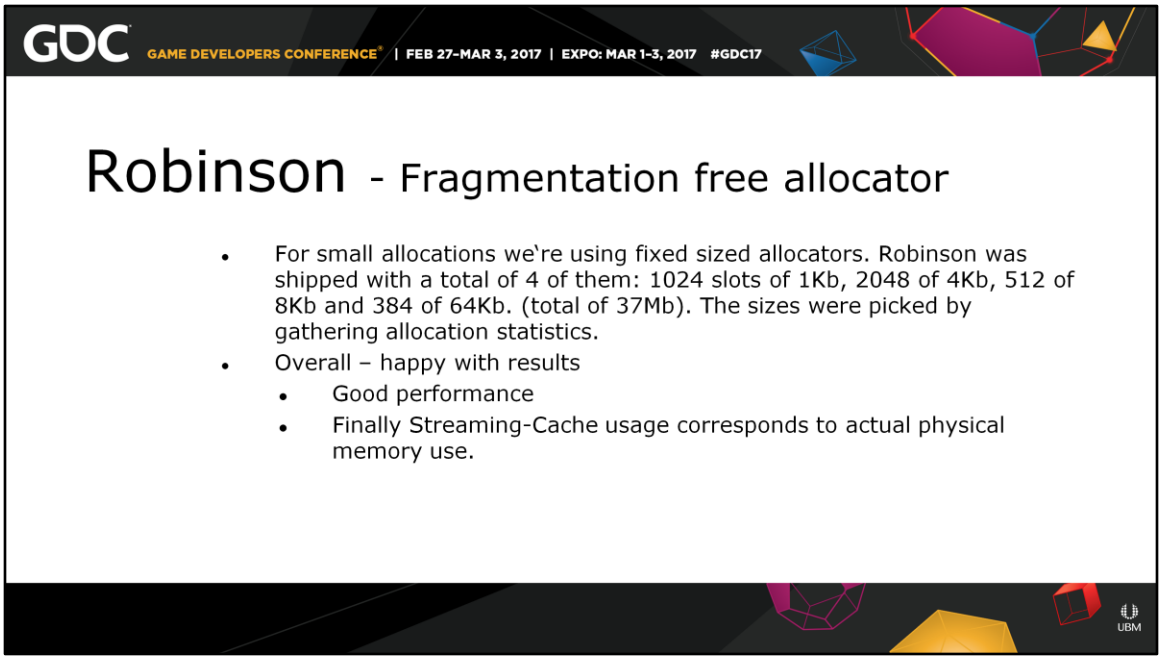
`<Details>`

a) On Allocate - after rounding up to requested page size multiple, we take the number of pages requested from the pool and map them as continuous in Virtual space. Order in physical space will quickly stop being continuous after a few rounds of allocate/deallocate – but hw only cares about what it actually sees – ie. The virtual range.

b) On Free – return the pages to pool and unmap them before any further use. Worth noting here:

a) Forced Unmap here uncovered a range of never before seen engine/game bugs where unmapped memory access could happen – ie. we were dealing with dangling pointers or other issues where the code assumed things about memory-state that weren't actually true. This was painful at first – but from perspective of sanitizing the newly adopted renderer-codebase, and just the general benefit of having this kind of „test" available for graphic resources, it was actually a nice side-benefit overall.

b) Sony API actually „used" to allow re-cycling pages without a specific unmap. Up to FW 3.0 (or thereabouts) it was possible to simply call map function and if the range overlapped with existing mapped pages, they would be remapped to the new values. There's obvious sinister implications of this API behavior – so it's understandable why it has been restricted, but from perspective of our application, it would be nice to get this back some day (save some performance too) – Sony, if you're listening, this would be really nice to have again ;)

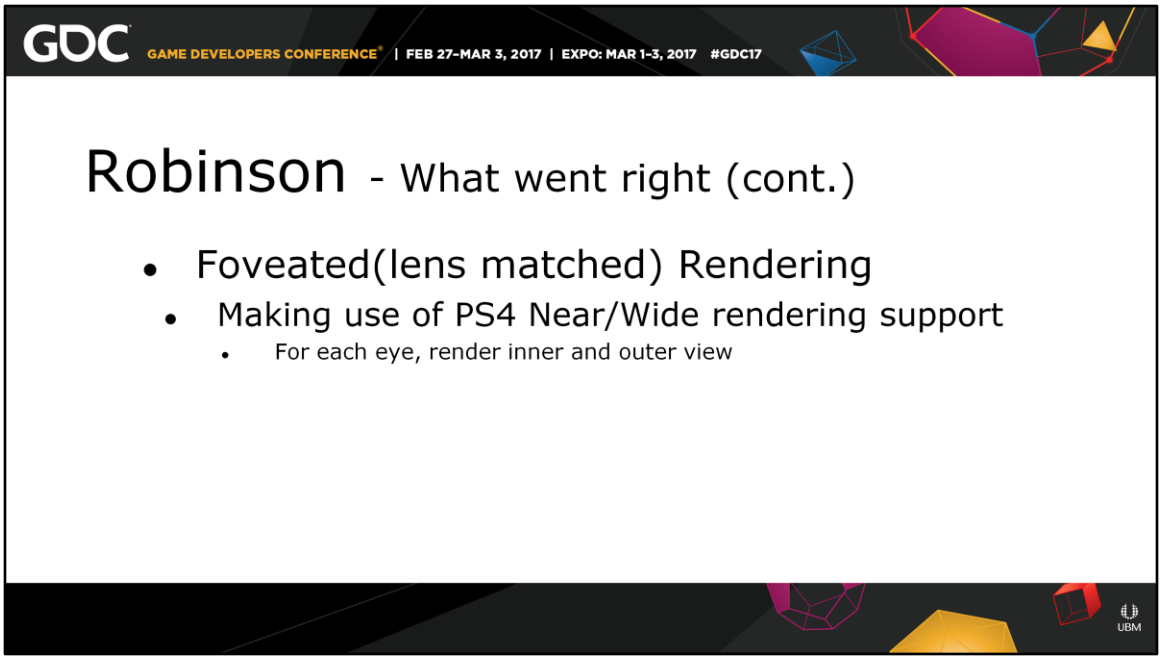


d) Allocation size handling. Obviously for anything above 64KB, page pool is used without problems – but smaller allocations would be too inefficient that way – so we've used a set of fixed-sized allocators to handle those – in Robinson that was 4 ranges 1KB, 4KB 8KB and 64KB, totalling just under 40MB. The settings for these were picked by doing a statistics run over all in-game allocation requests, so the said figure was quite optimal in terms of efficiency.

If a fixed allocators runs out of space – it will default to the next available higher range, until eventually falling back onto page-pool in worst case.

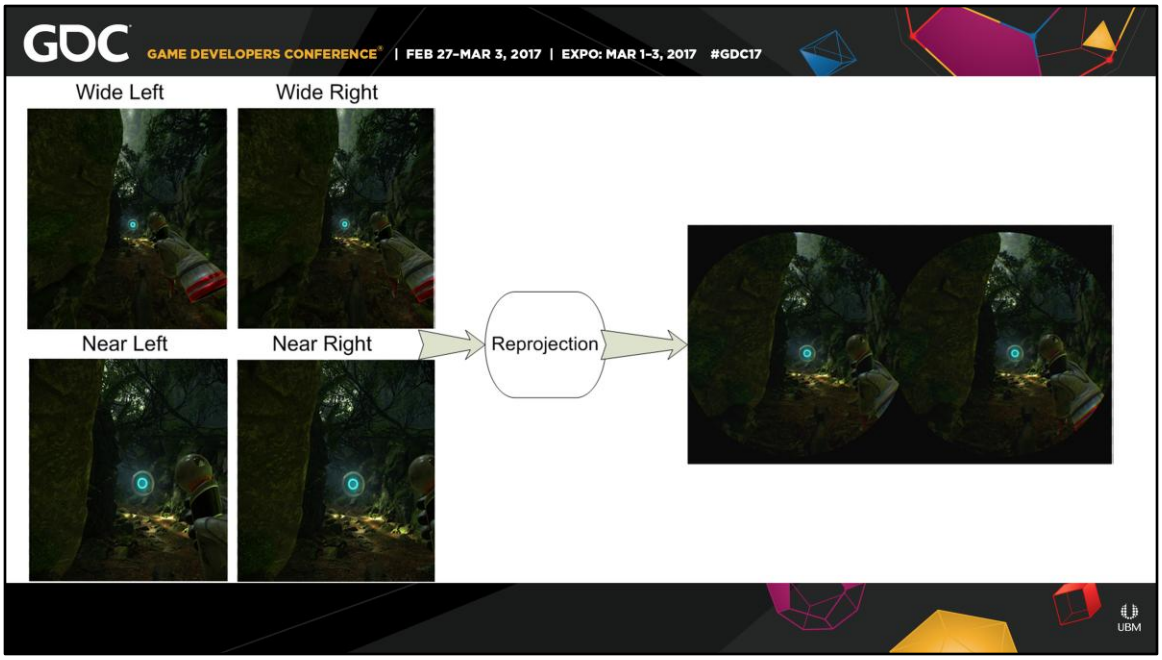
Overall results were quite good – runtime performance never became a problem (do take note that Map/Unmap calls aren't free, so these systems should be handled with care still), and having all our graphics-physical memory allocations now fragmentation free, removed a lot of headache in the final days of performance and asset tuning.

Obviously – there's always room for improvements.

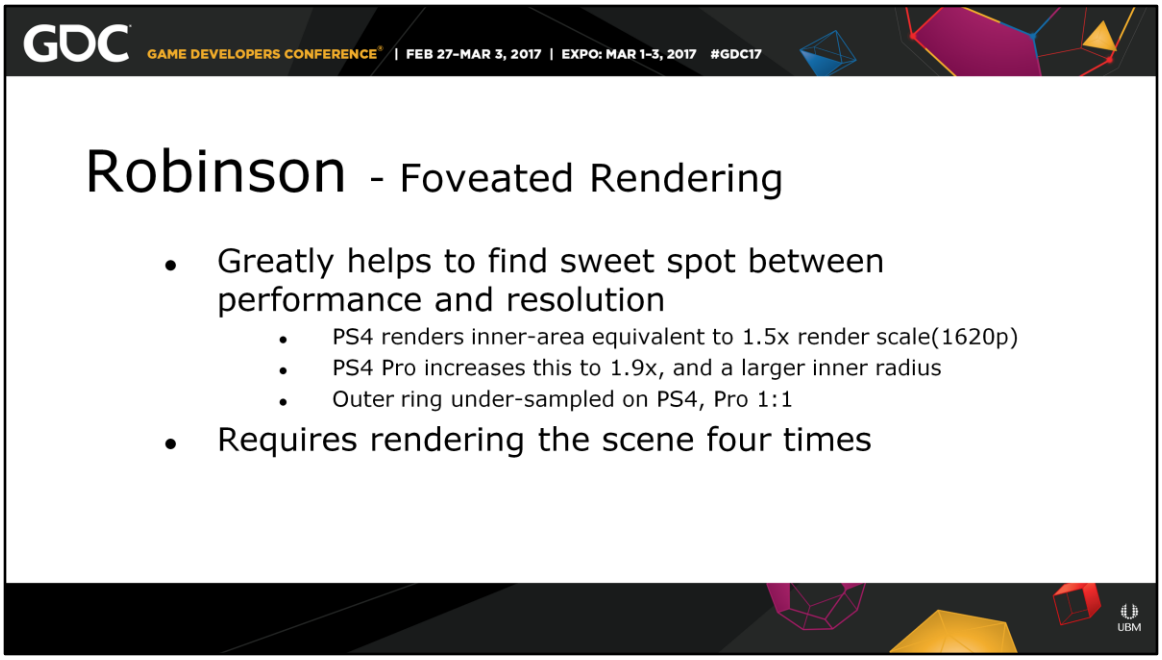


How we went from 1080p and 600 draw calls (with 2 camera views) to 1620p (1.5 render-scale equivalent) with up to 3600 drawcalls (4 views), while simultaneously reducing GPU budget and not increasing the CPU budget.

Implemented lens-matching to get our Pixel budget and quality up.



How we went from 1080p and 600 draw calls (with 2 camera views) to 1620p (1.5 render-scale equivalent) with up to 3600 drawcalls (4 views), while simultaneously reducing GPU budget and not increasing the CPU budget.



The ratio of near and wide regions is key.

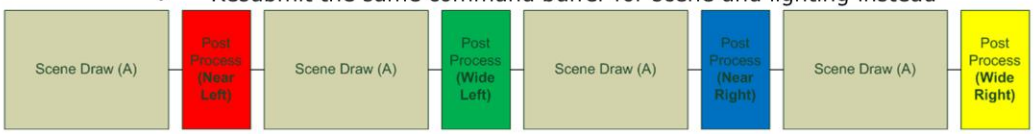
On PS4 we were slightly more aggressive – with near:wide field-of-view ratio being 0.4 / 1 – allowing for near circle hitting the sweet spot equivalent to 1.5 renderscale (slightly super-sampled pixels in the center). The cost of the ratio favoring the inner ring meant that Wide area is slightly undersampled, but accomodating for lens-distortion on the edges, the artifacts are minor for most part, and only noticeable when explicitly focused on. We did observe here that repositioning the near „circle“ lower to accomodate for pantoscopic tilt of human-vision would yield better results (we pickup artifacts better in the bootm part of vision than above – so pixel distribution should cater for that too). Blending region has been increased accordingly as result as well, to hide the transition artifacts better.

PS4 Pro expands the ratio 0.45 / 1, and roughly doubles the pixel counts, giving us equivalent of 1.9 renderscale in center (a nice supersampling amount) and bringing wide region sampling to 1:1 rate. There‘s still a quality discrepancy between the regions if you really pay attention to it(as center is supersampled), but it‘s very subtle at this point.

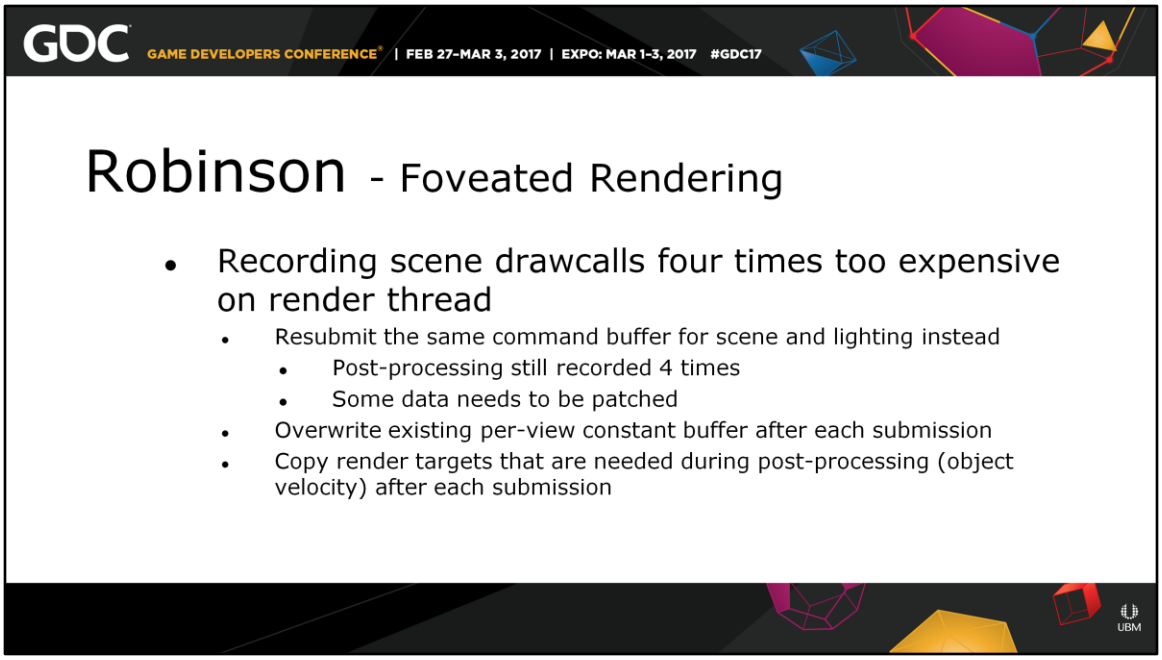
For most part – rendering pipeline translated well to this new setup – but there were a few challenges – most notably, in post-processing pipeline with effects that can sample in irregular patterns (like SSDO and Fog volumes) and others that had to be made „Near/Wide aware“ to generate correct results (Antialiasing and Lens-flare pass most notably). Shipping version resolved issues with all but one of these (eliminating transition artifacts) – SSDO was unfortunately not something we could fully solve – future work here would be to allow some effects to run over a merged-buffer to ensure correctness.

Robinson - Foveated Rendering

- Recording scene drawcalls four times too expensive on render thread
  - Resubmit the same command buffer for scene and lighting instead



Constant buffers and Htile masks are patched-per view for Scene-Draw, and we make copies of any render-targets needed by post after each submission.



How we went from 1080p and 600 draw calls (with 2 camera views) to 1620p (1.5 scale equivalent) with up to 3600 drawcalls (4 views), while simultaneously reducing GPU budget and not increasing the CPU budget.

Robinson - Foveated Rendering

- Saving GPU costs
  - Vertex shader executed 4 times but overhead was very acceptable
  - Absorb vertex-overhead in GBuffer by interleaving Post as an Async-Job
  - Populate HTILE mask to reject pixels that are outside relevant regions

Near/Wide buffers obviously have some region overlap – so we use Htile masking to reduce redundant pixel work.

<Future work>

Dynamically controlled buffer sizes

Const pixel density with dynamic foveation

Pantoscopic tilt-matching of FOV areas

How we went from 1080p and 600 draw calls (with 2 camera views) to 1620p (1.5 scale equivalent) with up to 3600 drawcalls (4 views), while simultaneously reducing GPU budget and not increasing the CPU budget.

Comparison between standard stereo-pass at 1080p – and Near/Wide optimized for Near target at 1.5x scale (standard PS4).

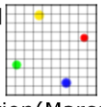
Significantly higher quality at a slight performance advantage.

Robinson - What went right (cont.)

- Temporal Supersampling
  - Image quality even more important in VR despite the high performance requirements
    - Aliasing very noticeable and disturbing
    - Any artifacts and lack of sharpness much more noticeable in VR due to observed pixel size
    - Preserving image sharpness essential for AA solution
    - Want and need the holy grail, perfect AA at no cost ☺

Better quality AA for less GPU time ;)

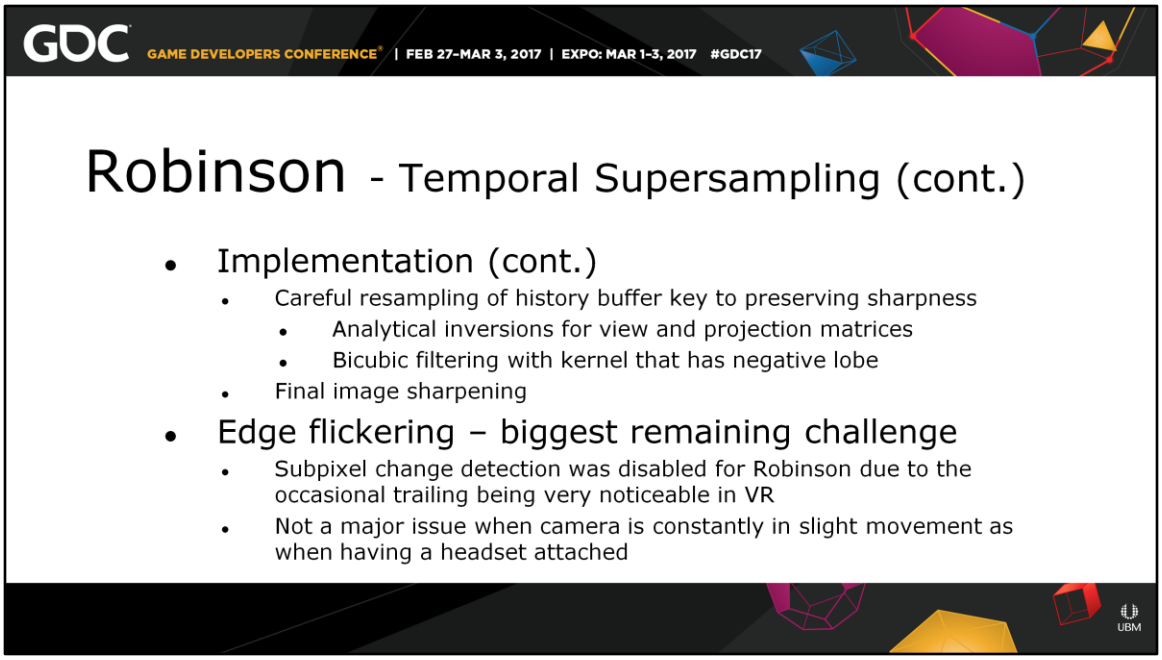
Robinson - Temporal Supersampling (cont.)

- Implementation
  - Temporal-only solution for performance reasons
    - Edge filter like SMAA orthogonal and optional
  - Use 4 samples ordered as a rotated grid 
  - Color variance clipping for history rejection(Marco Salvi – GDC 2016 http://www.gdcvault.com/play/1023521/From-the-Lab-Bench-Real)
  - Slight mip bias used during gbuffer generation
    - Look at costs, negative bias is quickly increasing bandwidth

Sharp temporal sampling implementation.

Temporal only solution to cut some of the costs – edge-filters like SMAA are orthogonal to temporal, and thus can be seen as optional. End results speak for themselves here of course ;)

Color Variance Clipping as described in last year's GDC talk from Marco Salvi - http://www.gdcvault.com/play/1023521/From-the-Lab-Bench-Real



Sharp temporal sampling implementation.

Careful with how we approach resampling of history buffer to preserve sharpness - Minimizing diffusion coming from floating point precision issues by using analytical inversions for view and projection matrices, all in double precision. Apply Bicubic filtering with kernel that has negative lobe (Catmull-Rom in our case)

Image sharpening – careful with usage - needs to be rather subtle to avoid overly noticeable ringing artifacts.

On Robinson guided by Art-director's "eyes" to ensure optimal results.

Edge-flicker solution was implemented – but we had to pass on it for Robinson in favor of avoiding any sort of motion trails. Less of an issue in VR anyway, as camera is constantly in slight movement.


GDC GAME DEVELOPERS CONFERENCE | FEB 27-MAR 3, 2017 | EXPO: MAR 1-3, 2017 #GDC17
Robinson - Conclusions
• Unstable plan/timeline
• "Late to the party" with development (performance, content, gameplay)
• Mixed results on locomotion and user options
• Highly successful with tech innovations
• Raised visual bar on the medium/platform
UBM

What have we learned

- VR is a new medium
  - Production realities and good practices still apply
  - Vertical slice (even for a public demo) can actually be an important tool for development if utilized the right way
    - Even when doing shorter-term productions that can be seen as experimental

VR being a new medium, we're dealing with very different set of restrictions and many established building elements of the games have to be adapted (or in some rare cases, even avoided). That being said, lessons around planning and production that our industry has struggled with for decades apply just as much as ever – even in this "experimental" phase of the medium we're currently in. In other words, production practices/processes are perhaps more important than ever – both projects in question highlight the importance of planning (in contrasting ways), as production team maintained a project plan against almost impossible odds in Robinson, which was a big part of why that title managed to ship.

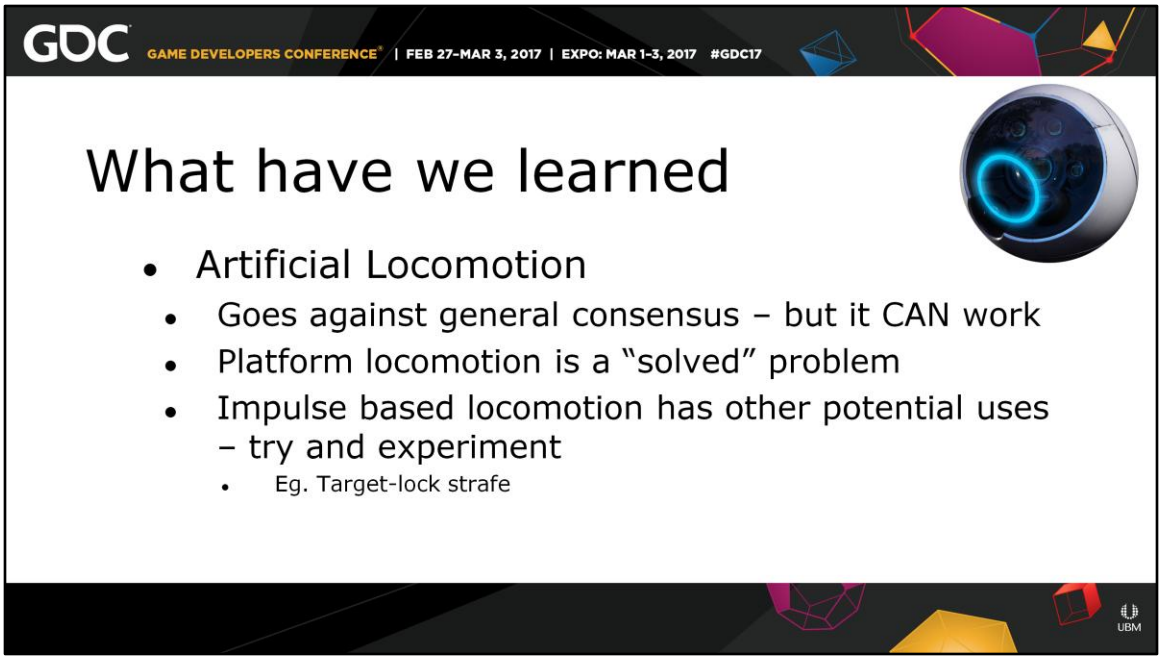
Process of course refers to more tangible elements as well – eg. performance is no longer just "good to have" but a fundamental element of the VR games, and maintaining a strict control over it requires co-operation from entire teams, and strong adherence to processes in each group.

Also, while I was never a fan of trade-show/public demos built purely for marketing purposes, if you manage to integrate them into your production pipeline, these can be an incredibly valuable tool for validating your tech and game concepts – especially when dealing with a medium that has no really useful established precedent out there.

What have we learned

- Novelty of the medium can work to our advantage
  - Less set expectations
  - Developing around highly polished core-concepts instead of pushing feature volumes works(still?)

The novelty of the medium isn't just a challenge to overcome though – it also works to our advantage. Innovation is encouraged, publishing partners are less interested in historical precedents of prior art, and customers themselves are more willing to try new experiences. This also lends itself more towards strong core-elements and highly polished experience around them - as opposed to pushing massive feature lists, which is also helped by current budgetary and business model/userbase constraints – at least for the time being.

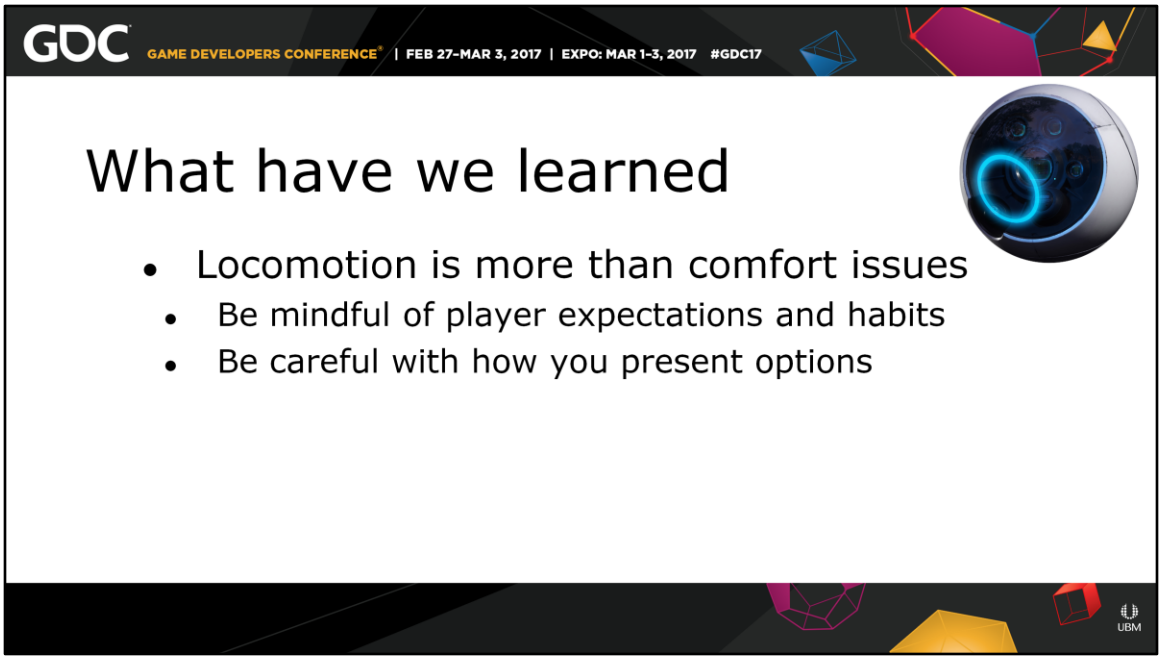


Artificial locomotion is another boogey man – and general consensus has been to avoid it or stick people into cockpits (which still has mixed results in many cases).

Climb proved us otherwise – without motion controls, it's universally accepted as a very comfortable experience – effectively platforming locomotion in VR is a solved problem(other examples out now). Motion controls make that somewhat moot(dragging the world has been concepted early on as acceptable by many/most) – but it's an important point as jury is still out on what input systems are here to stay in VR.

<Future stuff>

Extension from above is that impulse driven locomotion where acceleration/movement is performed relative to view-direction is in fact acceptable for most individuals – even when you mix in lateral and rotational motion. Orbital strafing/target-lock prototype (actually exists in Robinson shipping code – but hasn't been enabled) demonstrates this well – where most users deal poorly with strafing, especially at higher velocities, orbital generally alleviates most sources of discomfort. At the time of this writing we were not yet confident in implementation being intuitive enough to use though.



Robinson was a mixed success. Indications are that base locomotion has been “mostly” comfortable, although results were not nearly as consistent as with Climb (looking at higher %s of participants experiencing some form of nausea with default controls).

But player expectations are another story - many long-time players initially struggle with the idea of using head “aiming” over the all too familiar DS4 sticks. This in turn yields many-more choosing the smooth rotation schemes as well, which of course only tend to worsen the comfort of the experience for the said players, leading to frustration with the title overall.

Option labelling is subject for some discussion here – as not all platform holders are open to simply sticking a “for players immune to motion sickness” onto a button.

<Future stuff>

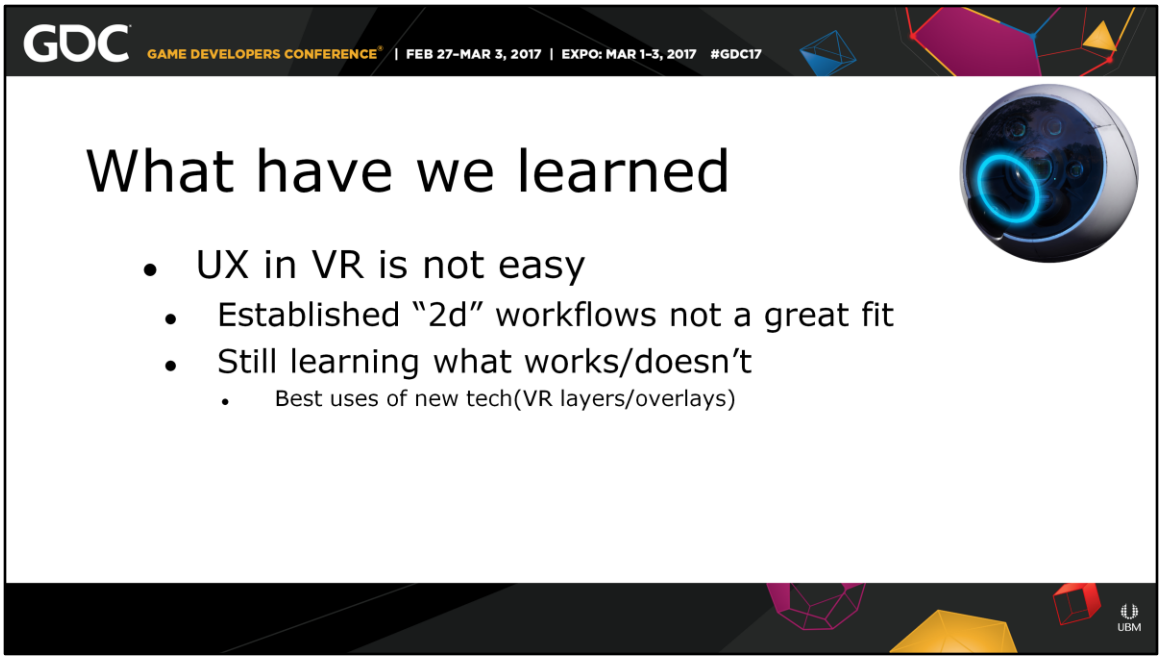
That said – this shouldn’t discourage you from further experimentation – Robinson also left many things on the cutting floor due to time constraints, and in the time since we’ve only grown in confidence in terms of what can be done in terms of free locomotion. There’s a wealth of options that work in different scenarios, it’s up to us(the VR dev community) to build it into a toolset that can be applied in general situations. The list below is only a sample of things that show potential – but generally you can think of it in two larger paths of research – comfort-assists/aids, and comfort-desensitization:

Per pixel motion flow control

Parallax layer UIs counterbalancing sickness triggers

Audio/Light grounding to tracking space

Desensitizing the user through unexpected



UX in VR is hard – or at least, harder than in 2d, if you want it done right, especially since existing workflows weren't built for it.

Moving things into 3d space is almost inevitable, but we're still learning to what extent and where 2d is still applicable and/or more useful.

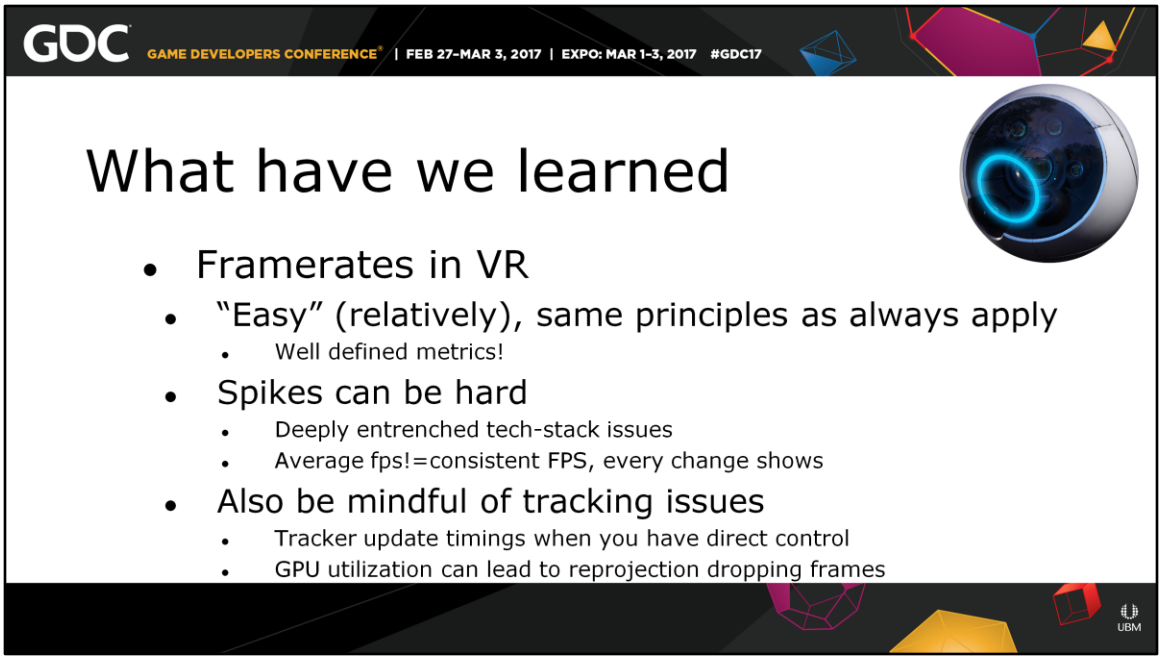
<Future stuff>

Beyond that – we always have to keep an eye on readability and performance. Layers(or Overlays, for OpenVR users) are supported by all major VR SDKs to date, and they offer a great way to provide game-render update independent UI – both in terms of resolution and framerate.

The latter is particularly important to titles that reproject at non native framerate (eg. 60->120) as this will result in visible judder in any headlocked UI element – unless you place it on a layer that doesn't do reprojection, or alternatively refreshes at 120hz clip already.

There's more interesting options if you move into 3d layers and your rendering tech supports them well – consider rendering cockpit at a different rate from the world for instance, and that's just stating the obvious example again…

Resolution independence (or high quality sampling path, for SDKs that support it) is another important feature – as it allows you to maintain UX always sharp and readable no matter what goes on in the game world (be it temporal, adaptive, or just plain-low spec rendering that doesn't guarantee good clarity) – basically beyond UI, layers also point to a way forward with asymmetric rendering/update rates across the board.



Framerate is the big boogey man in VR, as it's no longer just a "nice to have" but an integral part of the experience. Dropped frames aren't an inconvenience, they're comparable to bad input sampling or screen-sized flashes of random memory – they take the user out of experience, and potentially worse (causing discomfort etc).

That said – maintaining a stable target framerate isn't exactly new, we've always had the tools at hand, it's just that the industry collectively decided to care less about it at certain points in history. Going by the learnings of the two titles described – hitting targets early and sticking to them obviously works best – it's not an optimization decision, it's a design one. Brute-forcing can still work for some things, as evidenced in Robinson – but even there, the measures put in place later were accordingly stricter to get us there.

What cannot be over-emphasized is the importance of good metrics for teams to work with – anything from scene-complexity, draw-call counts, to specific GPU subsystem budgets, profile early, and set clear guidelines for your team, along with optimization recommendations for different cases. While it's frequently seen otherwise from programmer side, content team can be your biggest ally in optimization if they are given the right set of rules(and tools) to work with.

What did prove to be more difficult – was tracking down the infrequent spikes – modern game codebases tend to be complex, with lots of difficult to track and sometimes unforeseen interaction possibilities between systems, and these can come to haunt you when you least expect them. What in a non-vr game would just be a blip that you "live with" – in VR becomes the aforementioned game-breaker. Profiling tools that can such track-spikes used during Robinson proved particularly important here, and PS4 Platform utilities were valuable as well (TRC notifications really help to keep discipline for the entire team). Finally be mindful of consistent frame-delivery, average of 16ms doesn't mean much if you frequently fluctuate above/below due to pipeline behavior.

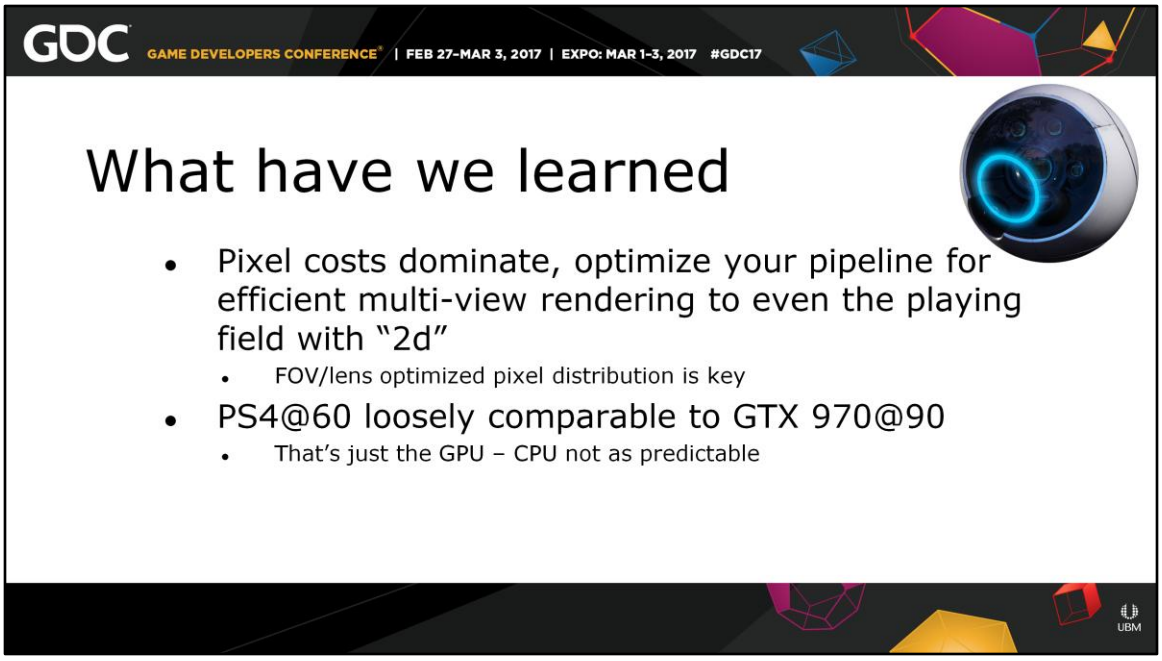
There's also the issue of tracking – that is usually expected to be the responsibility of SDK – but can be a source of problematic issues on some platforms. Keeping latency down is one such problem – as we discussed earlier in presentation, but there's also the issue of dropping tracking-frames or reprojection frames due to overall system load, which look just as bad(or worse) as frame-drops to the user. We've literally encountered that on Robinson where late optimization increased GPU utilization to the point that reprojection-job started dropping frames, forcing us to chose less aggressive timings on it.

On PC there's precious little control you get over that, but on consoles it's entirely the responsibility of developer to address those (which can be trickier if you work with 3rd party middleware).

What have we learned

- High-fidelity on current gen-hardware
  - Very achievable
  - Even staying deferred and dynamic
  - Main concessions were shadow-casting light usage and post

Worth point out that staying with deferred shading and a completely dynamic lighting model, we were still able to hit our visual targets – and while forward pipeline was considered at one point, it resulted in little difference in tests performed. The largest sacrifices came in quality of some of the post-processing, and the extent of use of dynamic shadows, which was strictly budgeted. Amusingly some of the concessions with post-process we made during Climb were “rectified” in Robinson by the virtue of gaining more usable GPU time through Lens-matching and Async-Post.



The world of high-end visuals today is mostly pixel-processing bound, but optimizing for lens-distribution and multi-view can bring costs closer to that of "flat" games.

When it comes to cross-platform considerations – we've obtained roughly comparable rendering workload from PS4@60hz as GTX970@90hz – before factoring in FOV optimizations.

That said – this is strictly a GPU workload comparison – and achieving the same parity on CPU side may present altogether different challenges(it did for us) depending on the codebase you're working with (as well as rendering API, target PC CPU spec and more).

In summary

- Artificial locomotion is here and to stay
- We have a long way to go with UI/UX
- VR Performance isn't hard(er) – spikes can be
- High-fidelity on mainstream hw is a reality
- We're just scratching the surface so far…

Credits & thanks

Co-authors of this talk:

Nicolas Schulz - Lead Rendering Engineer, CRYENGINE

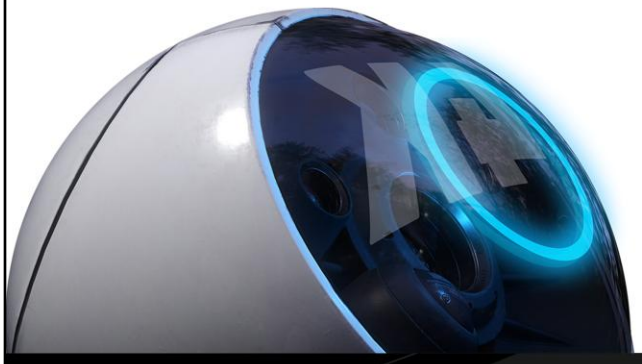
Matthijs van der Meide - Senior Engine Programmer, CRYENGINE

Andriy Pivnenko - Senior Systems Programmer

Dario Luis Sancho Pradel – Lead Game programmer

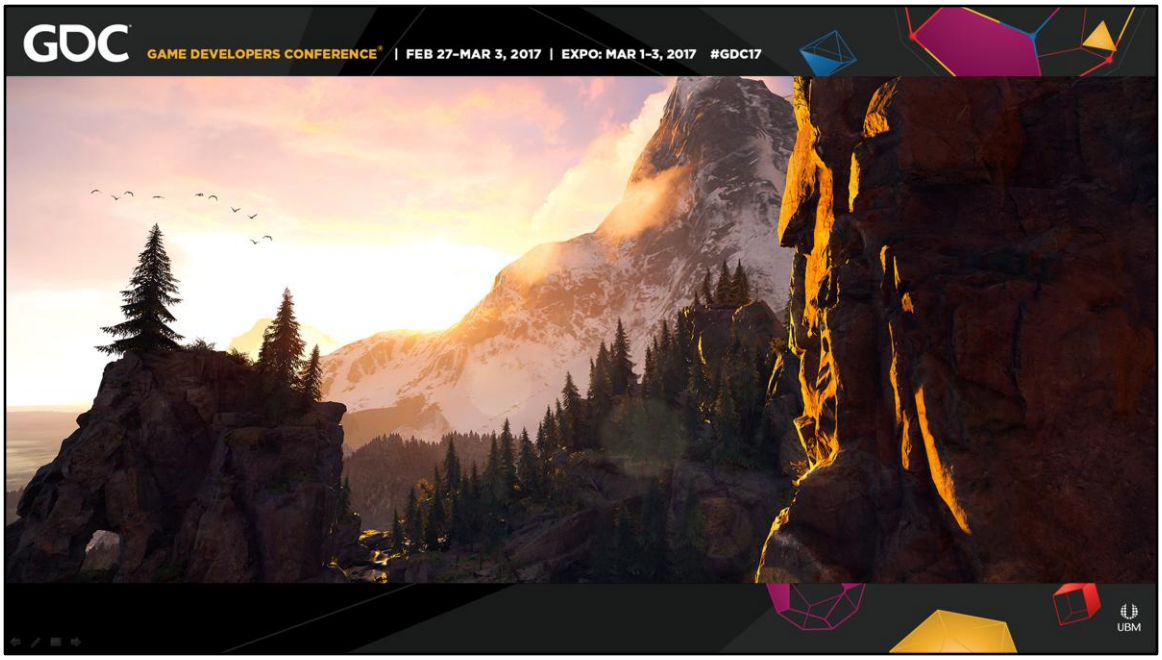
Special thanks to Robinson and Climb development teams for making the last 2 years a success!

Questions?



Bonus Slides

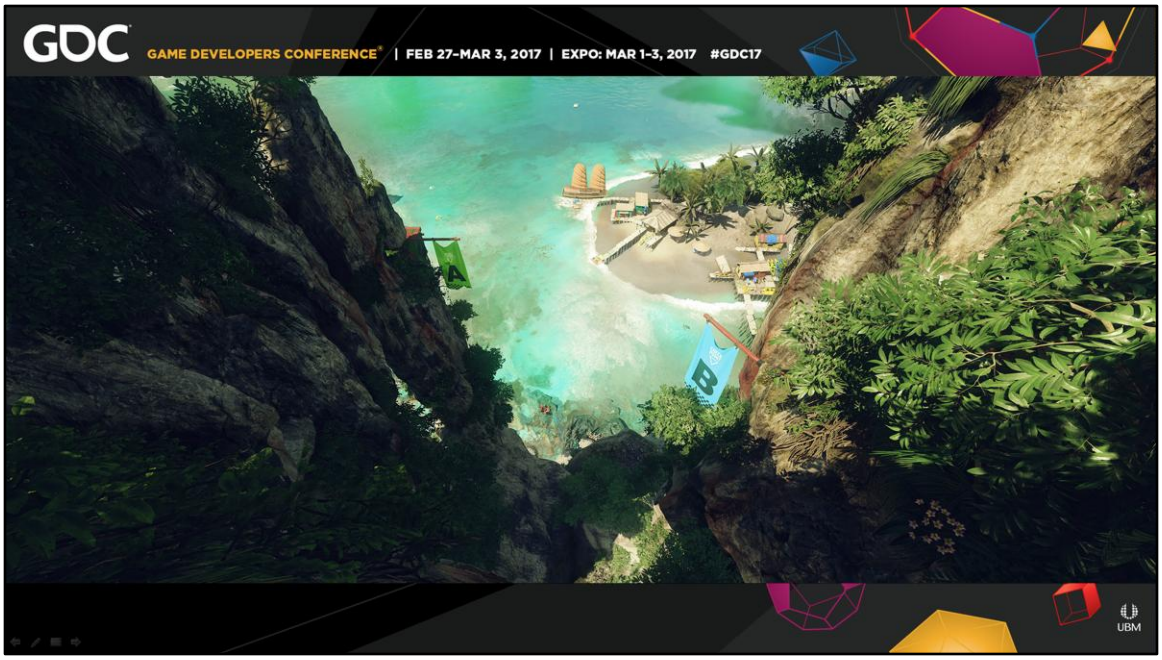
- Tech details that didn't quite fit into 60min



Vistas in Climb are temptingly like sky-box at first-glance, but movement verticality highlights massive amounts of parallax in the surrounding scene, so actual geometry was preferred for entire level. This in turn requires lighting and shadow for all of it – and at 50km, we've used up to 8 cascades to cover the terrain (a fairly standard LOG2 setup with "fair"-precision shadows in near-cascade of about 3m, 1024x1024).

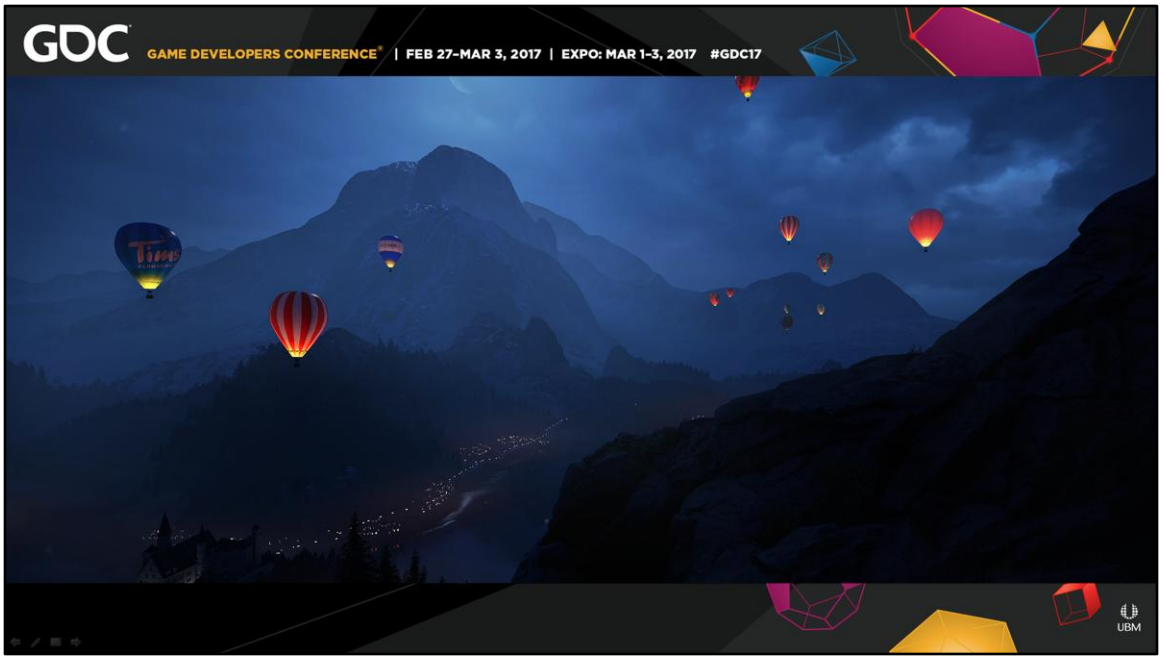
Rasterization of such distances turned out fairly CPU limited affair, which is where cached shadow-cascades were used to improve performance.

This in turn meant we had to light and shadow them in runtime – as in CE we don't actually do any sort of baked solutions to illumination, and it's not really in Crytek DNA to accept shortcuts for this. If that wasn't bad enough – as the old saying goes "give the art-team a finger and they'll take off your hand" – the lighting design for the 15 levels shipped in release version of The Climb actually called for over-dramatic, near-horizon sun location at least once per setting, generating a nice worst-case scenario for runtime shadow-maps.



<4 horsemen of shadow panning issues> Visually the bugs reported here looked very similar, but came from 4 different sources:

- Missing Shadow-cascades - simply not covering enough of the terrain with shadows, in final game we settled on 8 cascades for levels that needed it
- Cached shadows missing casters - the nature of how shadow-cache works imposes some limitations when dealing with very LARGE casters – again a common thing in Climb levels. There was no complete fix for this, but a hack where we pancake anything in a shadowmap that falls in front of near plane to near plane worked really well to mitigate it.
- Sun Shadows Clip Plane Range – ie. the value controlling the distance between shadow frustum near and far planes. Extreme positioning of the sun (near horizon) forced us to raise this value a lot in certain levels to “fit” the casters into the volume, trade-off being obvious precision sacrifices and potential increase in draw-calls (though the latter is mitigated by relying on cache for all our far-cascades).
- Frustum optimizations – tying the camera-pose to shadow-frustrum corner to get optimal resolution is sadly pretty terrible in VR, where the camera/head is in constant motion, revealing a very “wobbly” look to shadows (and the VR world itself) as result. Making the player-position(and disregard head-tracking) the center of cascade frustrum instead (at the cost of roughly halving precision) resulted in a much more pleasing motion-stability, which in VR makes the difference in selling the world as believable.


GDC
GAME DEVELOPERS CONFERENCE
| FEB 27-MAR 3, 2017 | EXPO: MAR 1-3, 2017 #GDC17
UBM

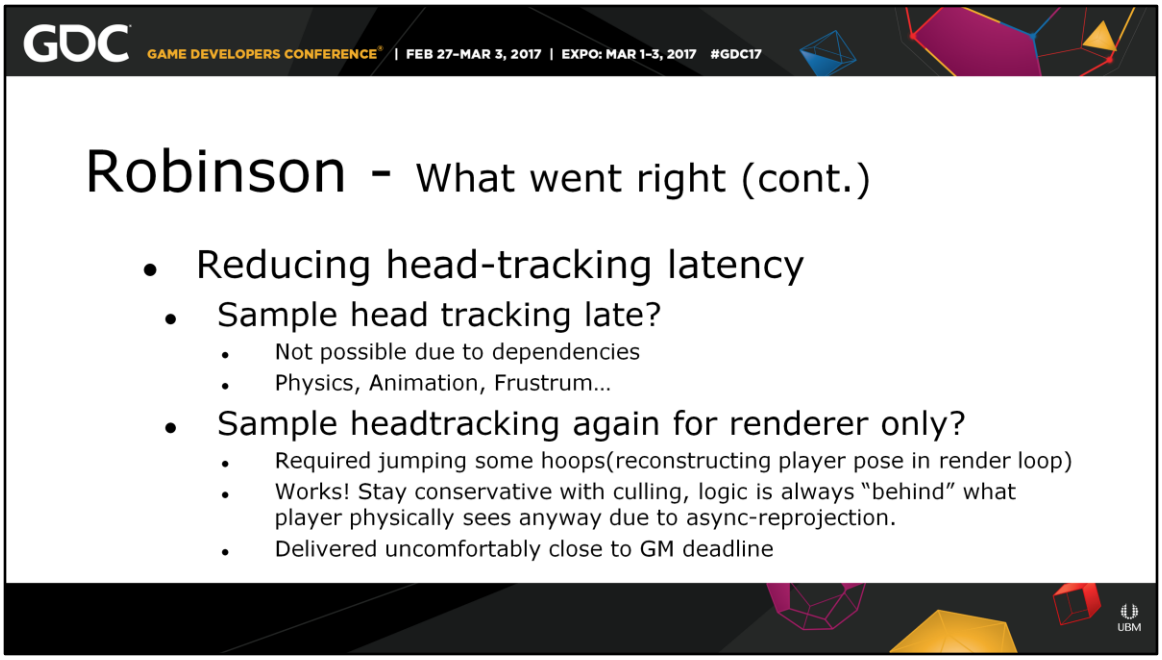
Robinson - What went right (cont.)

- Reducing head-tracking latency
  - Modern engines are pipelines
    - CryEngine gives default latency of ~3 (standard double buffered output)
    - HMD tracking optimal between 20-30ms of predicted latency, 60fps~45-50ms -> Leads to jitter at micro-scales, especially in large levels.

Before(~50ms) After(~30ms)


Modern engine pipelines typically consist of a literal pipeline where several large worker-threads(or groups of threads/jobs) produce different work-stages allowing for maximizing parallelism at the cost of extra “motion-2-photon” latency for every added stage. CryEngine default configuration has a latency of 3 – with main/logic stage - doing the obvious, rendering stage producing command-buffers, and hw/driver thread submitting to GPU (though on PC, depending on the vendor, you may complete rendering the frame in less than 3 frames from start to end thanks to immediate-submit modes if driver permits it).

While the total latency will vary depending on the software stack in use, there’s always going to be “some” – and HMD tracking APIs provide prediction facility allowing the application to absorb the processing latency to a degree. As you increase the prediction latency, tracking accuracy begins to drop – with recommended values being below 20-30ms for most APIs. In Robinson we were looking at 45ms+@60fps, and while the up to 50ms of prediction didn’t immediately manifest itself in errors, the combination of larger than usual worlds (due to some design decisions - Robinson playable world begins at nearly 1km off origin and covers multiple square kms), and first person VR allowing player to observe micro-detail at any point in the game, we eventually observed that head-tracking was noticeably jittery when putting your head next to a telescope, for instance.



Head-tracking was sampled relatively early in logic-stage, but unfortunately there was no easy way to move it – due to dependencies down the chain (animation, physics, actual logic execution, scene-culling). Instead, we decided to update with fresh-head pose only for the rendering-camera update (injecting it early in the rendering-stage, saving almost an entire frame worth of latency, bringing our prediction latency below 30ms consistently. This required jumping some hoops (re-run part of player-logic to reconstruct the correct head-pose from new HMD pose), but it ultimately turned out less problematic than anticipated.

This does mean game-logic "sees" a different position of head than the renderer does – but for most part differences will be equivalent to the now eliminated micro-stutter, and on any platform that performs async-reprojection in the final stage like PS4, you're already accepting this fact as default state anyhow. You should definitely take care to make any view-frustrum dependent logic (like culling) more conservative to account for that (chances are you do that already if reusing frustrum-cull in both stereo-views for instance).

Robinson - Temporal Supersampling (cont.)

- Edge flickering – biggest remaining challenge
  - Subpixel features that are just visible in one of the jittered frames can get classified as invalid by history validation
  - Considerably mitigated by looking at neighborhood in history buffer
    - Clamp individual history neighborhood pixels against color window and find color difference
    - Sum of differences used to determine if a pixel should be accepted
    - Can create new artifacts like trailing for single-pixel features

Sharp temporal sampling implementation.

Robinson - Temporal Supersampling (cont.)

- Edge flickering – biggest remaining challenge
  - Flickering mostly noticeable when camera is standing still
    - Not a major issue when camera is constantly in slight movement as when having a headset attached
    - Subpixel change detection was disabled for Robinson due to the occasional trailing being very noticeable in VR

Sharp temporal sampling implementation.

Robinson - What went right (cont.)

- Lighting optimizations
  - Wanted to continue using dynamic lighting system
    - Compute shader based tiled shading performs well on GCN architecture
    - Needs to be optimized as much as possible for 60Hz stereo

Lighting system improvements and optimizations.

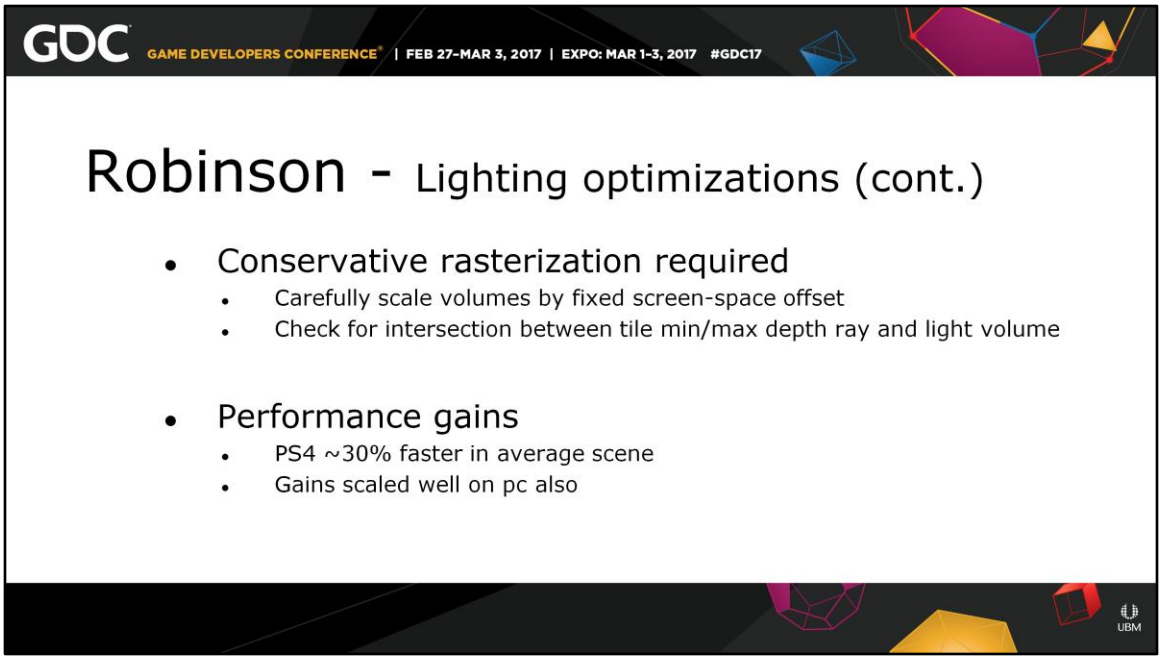
Robinson - What went right (cont.)

- Volume-based tiled shading
  - Rasterize light volumes to tile buffer (using UAV output)
  - Less false positives than frustum-based light culling
  - 2 drawcalls for each volume type: inside and outside geometry

Lighting system improvements and optimizations.

Within each volume type drawcall – overlapping volumes can have race conditions, as environment probes require deterministic order.

Use light bitmask to make results order-independent and deterministic, using Interlocked-Or on mask for each visible light.



Lighting system improvements and optimizations.