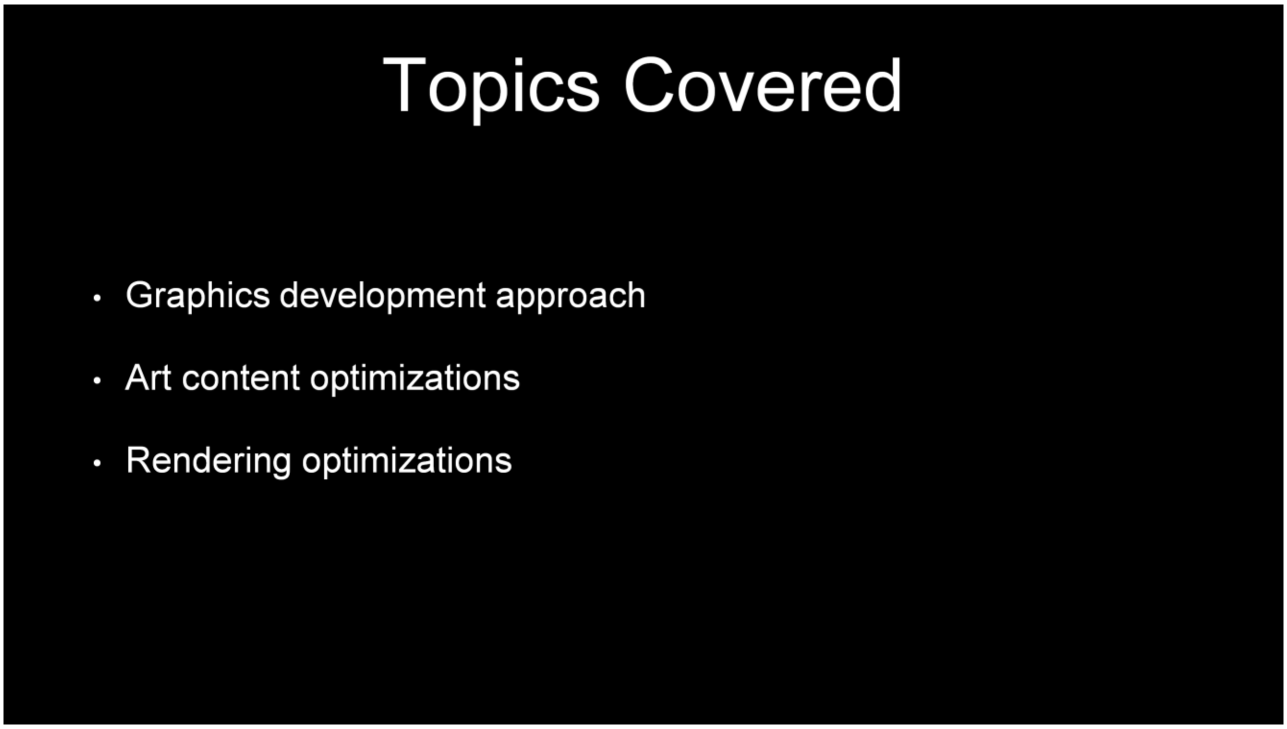
EVE: Gunjack is an arcade turret shooter, built from the ground up for VR, and is CCP's first mobile VR game. Gunjack was released in Nov 2015 alongside Samsung's first consumer version of GearVR and became a top seller on the Gear VR.

Play Gunjack Trailer -- 1 min 16 sec

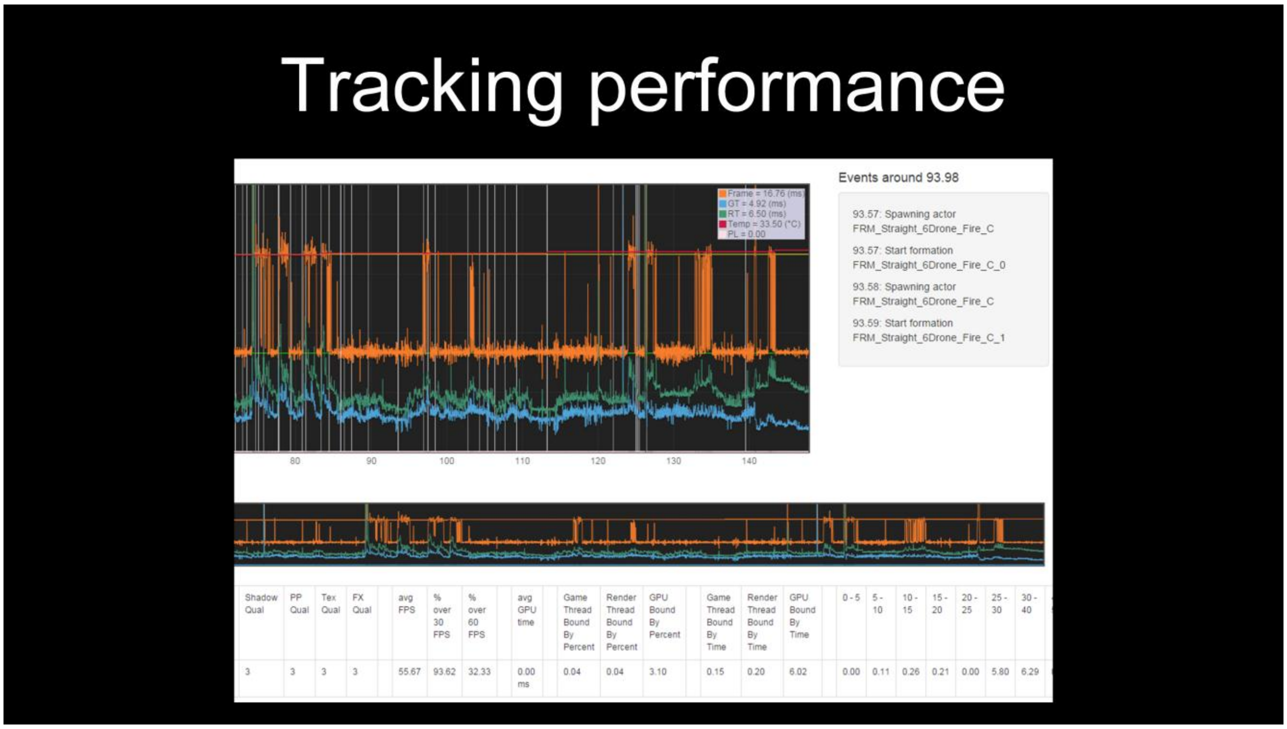


Topics Covered

- Graphics development approach
- Art content optimizations
- Rendering optimizations

Intro slide to what main areas will be covered.

Biggest challenge in Mobile VR is performance – for good player experience it required to render at 60 frames per second to avoid VR sickness and at high resolution to reduce the effects of Screen-dooring.



Tracking performance

Events around 93.98

93.57: Spawning actor FRM_Straight_6Drone_Fire_C

93.57: Start formation FRM_Straight_6Drone_Fire_C_0

93.58: Spawning actor FRM_Straight_6Drone_Fire_C

93.59: Start formation FRM_Straight_6Drone_Fire_C_1

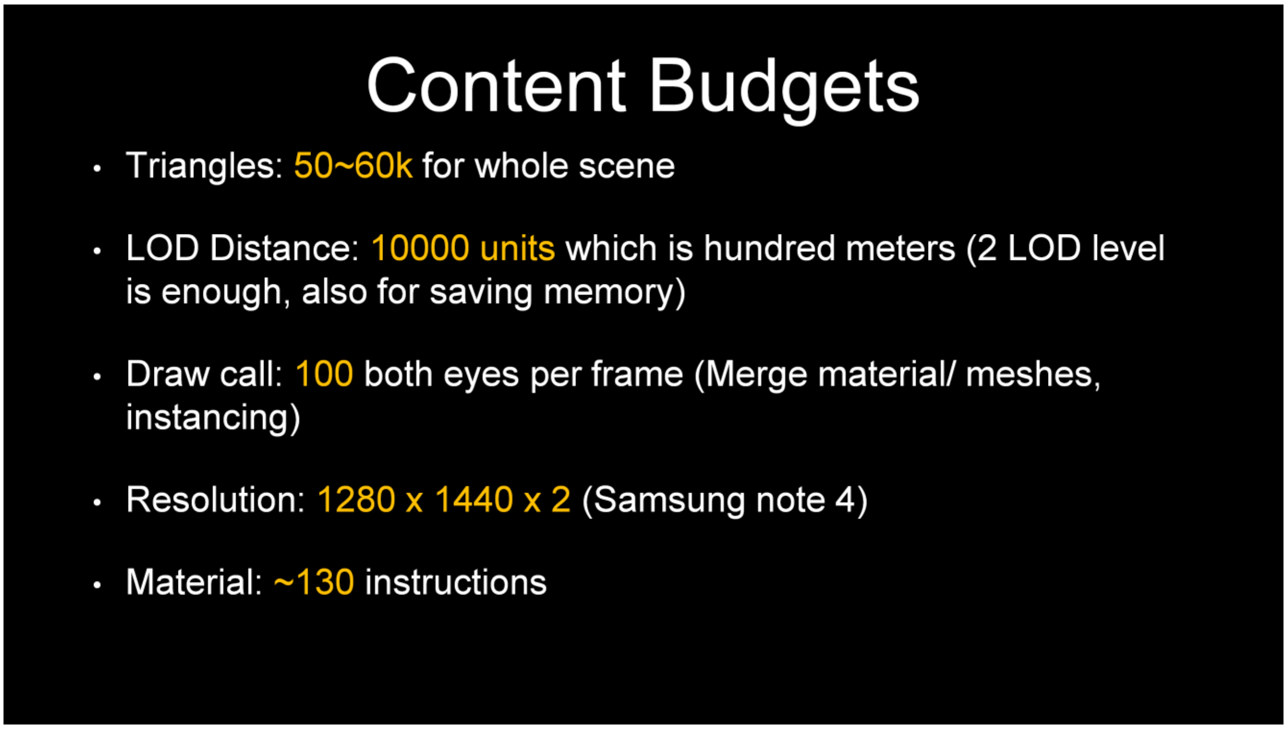
Shadow Qual	PP Qual	Tex Qual	FX Qual		avg FPS	% over 30 FPS	% over 60 FPS		avg GPU time		Game Thread Bound By Percent	Render Thread Bound By Percent	GPU Bound By Percent		Game Thread Bound By Time	Render Thread Bound By Time	GPU Bound By Time		0 - 5	5 - 10	10 - 15	15 - 20	20 - 25	25 - 30	30 - 40
3	3	3	3		55.67	93.62	32.33		0.00 ms		0.04	0.04	3.10		0.15	0.20	6.02		0.00	0.11	0.26	0.21	0.00	5.80	6.29

Screenshot of a tool that allowed the whole team to monitor and optimize performance.

Specifically useful is the white lines that show game events – this allows designers to not overload missions and cause bad performance

Also note that that the charts show

device temperature – more about this in slide 6

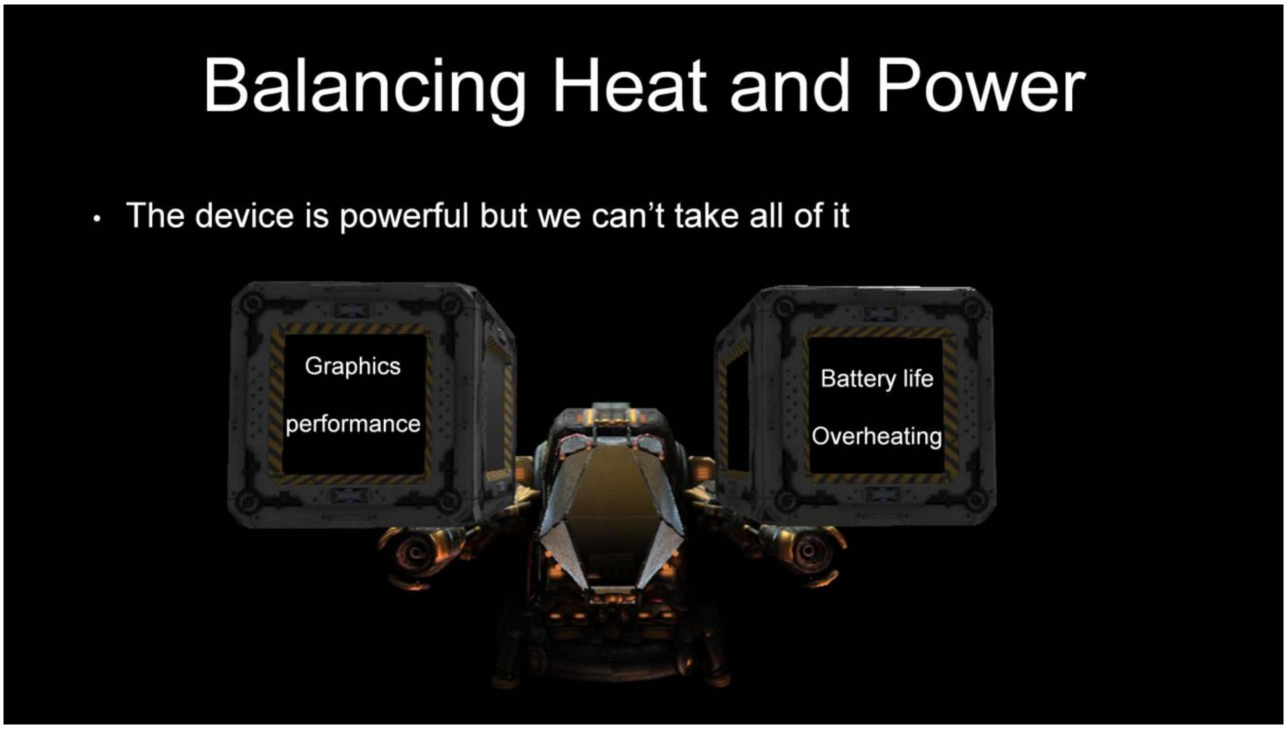


Content Budgets

- Triangles: 50~60k for whole scene
- LOD Distance: 10000 units which is hundred meters (2 LOD level is enough, also for saving memory)
- Draw call: 100 both eyes per frame (Merge material/ meshes, instancing)
- Resolution: 1280 x 1440 x 2 (Samsung note 4)
- Material: ~130 instructions

Alongside performance monitoring tools, we also created content budgets – these took some iteration and this shows what we used to make Gunjack – hopefully useful to you guys.

Particularly restrictive is draw call count.



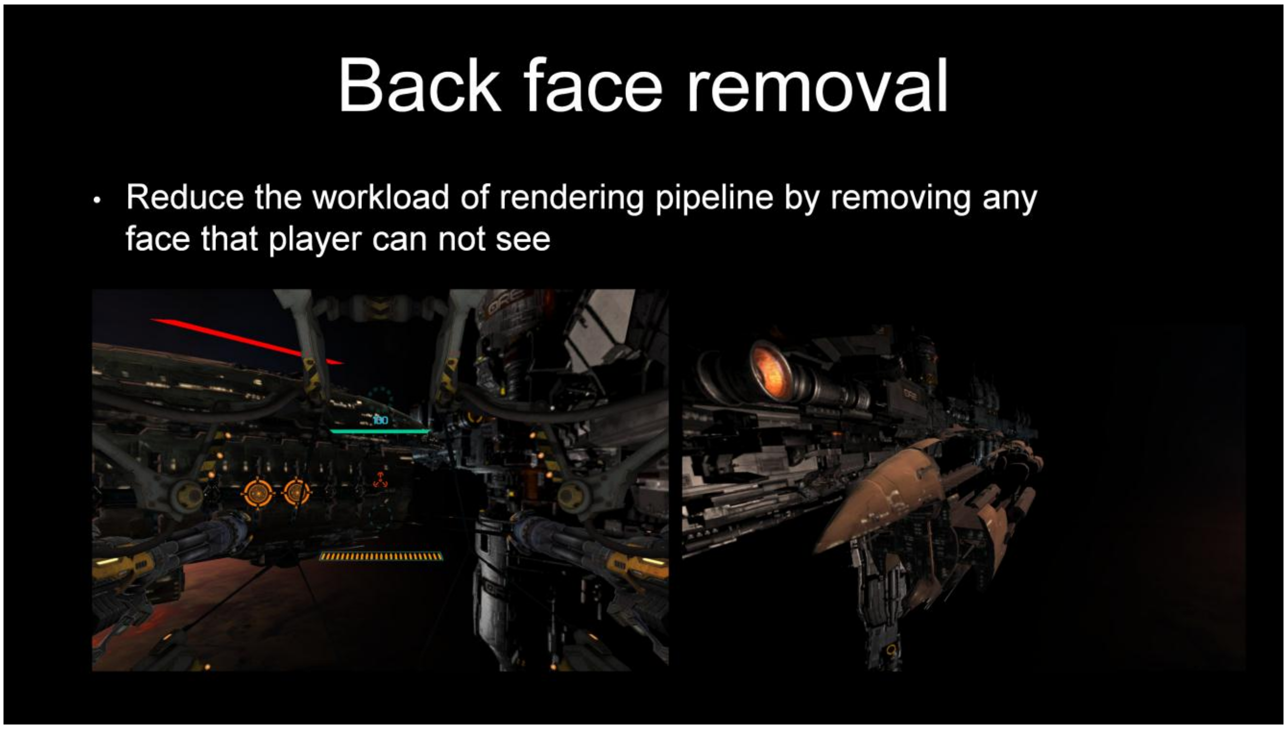
Balancing Heat and Power

- The device is powerful but we can't take all of it

Graphics
performance
Battery life
Overheating

Problem: running device at full clock rate causes device to overheat in a few minutes.

Solution: don't run device at full clock rate most of the time.

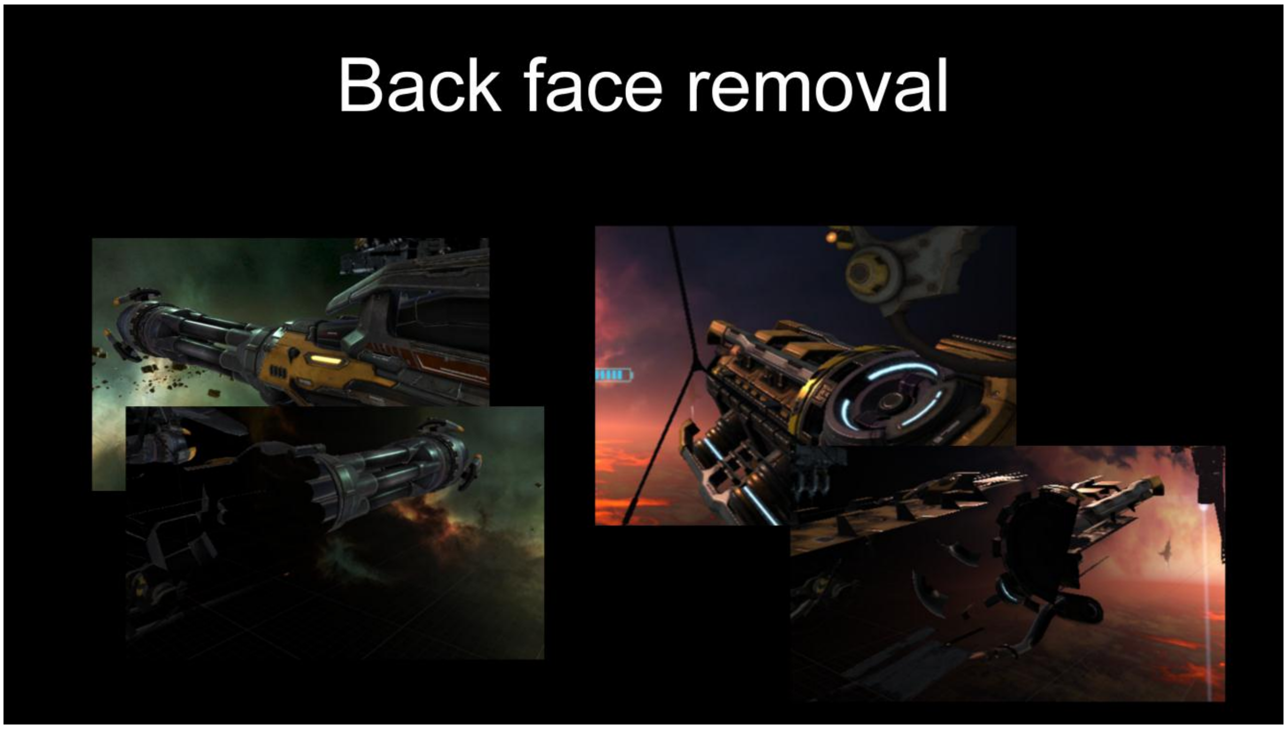


Back face removal

- Reduce the workload of rendering pipeline by removing any face that player can not see

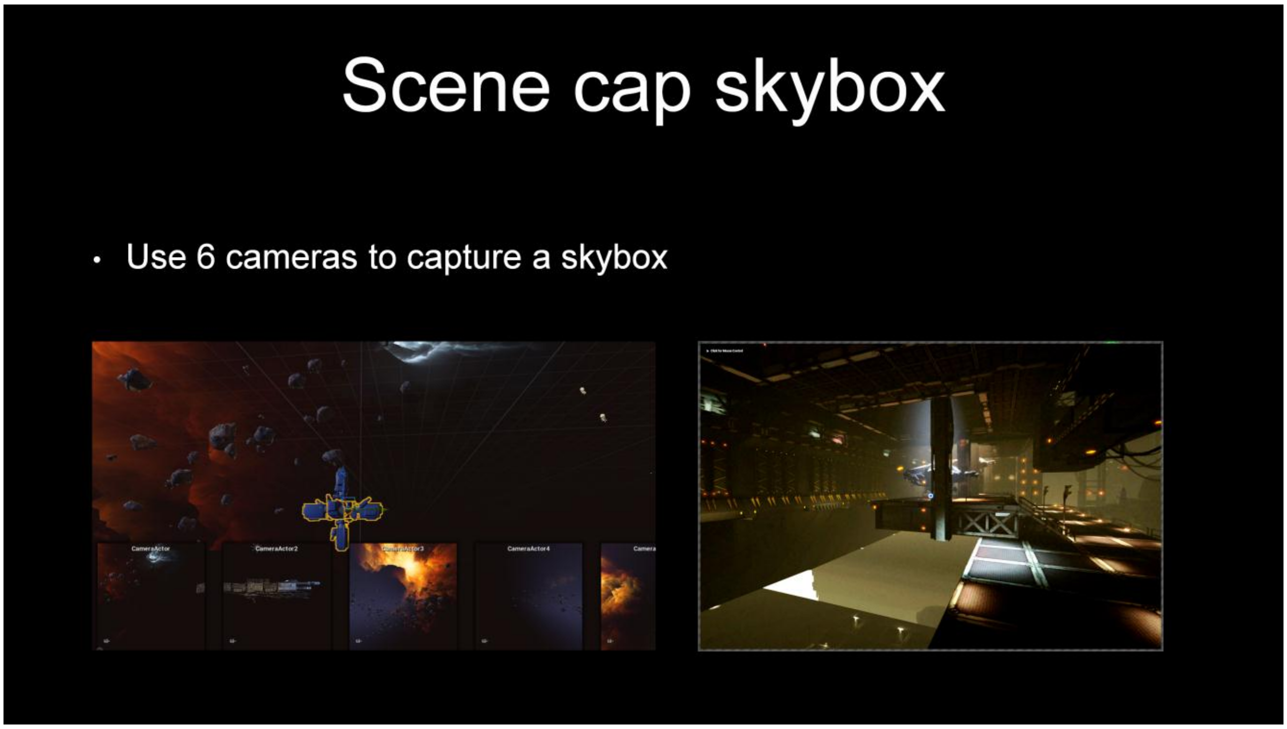
An important content optimizations we made in Gunjack.

Not a new technique but does have an impact – don't waste memory or bandwidth on verts/faces that the player just won't see.



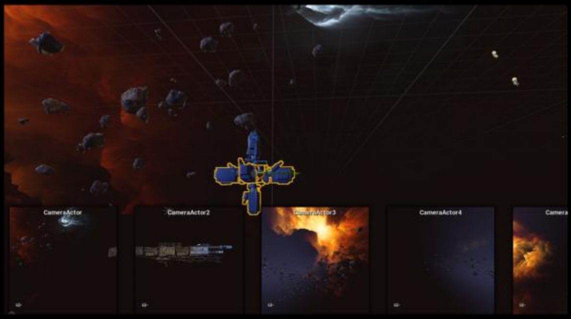
Back face removal

More examples of removing back faces that just cannot be seen by the player – these turrets always looks solid from players point of view.

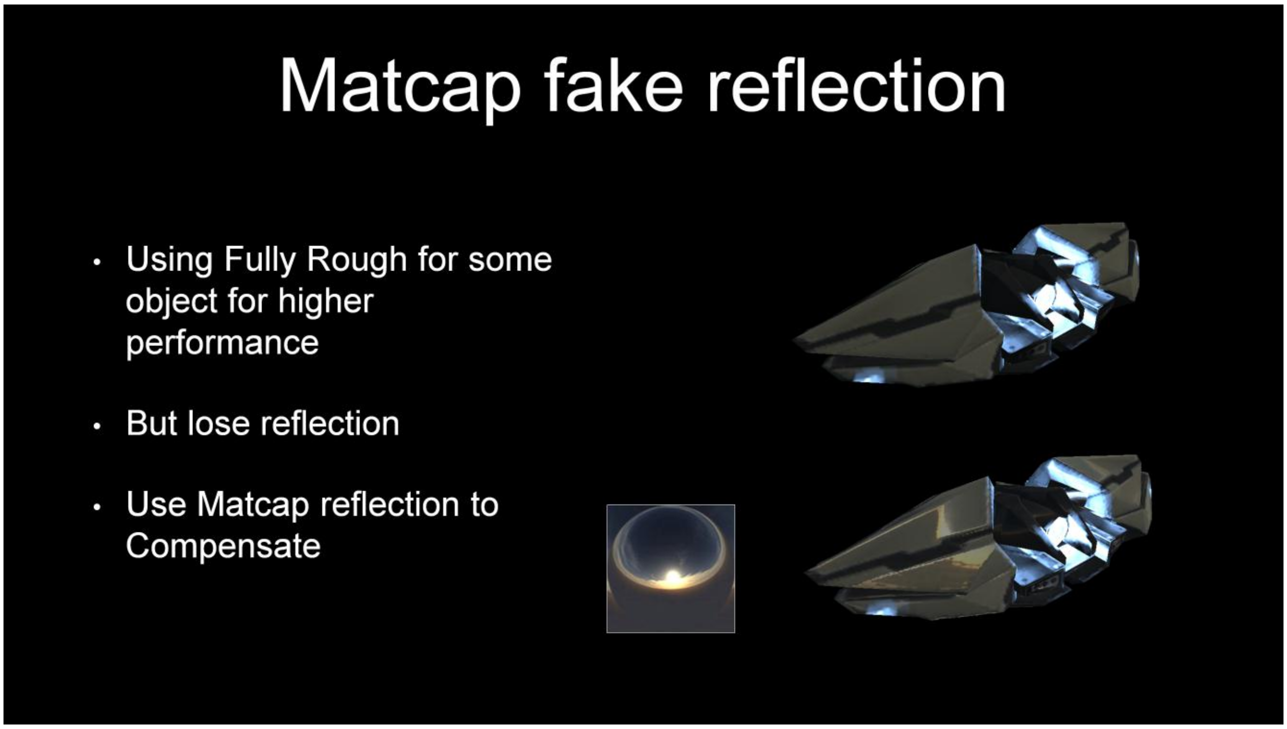


Scene cap skybox

- Use 6 cameras to capture a skybox



Reduces background scene to just a pre-rendered sky box – used in combination with some clever UV and vertex colour and composited with a detailed 3D foreground



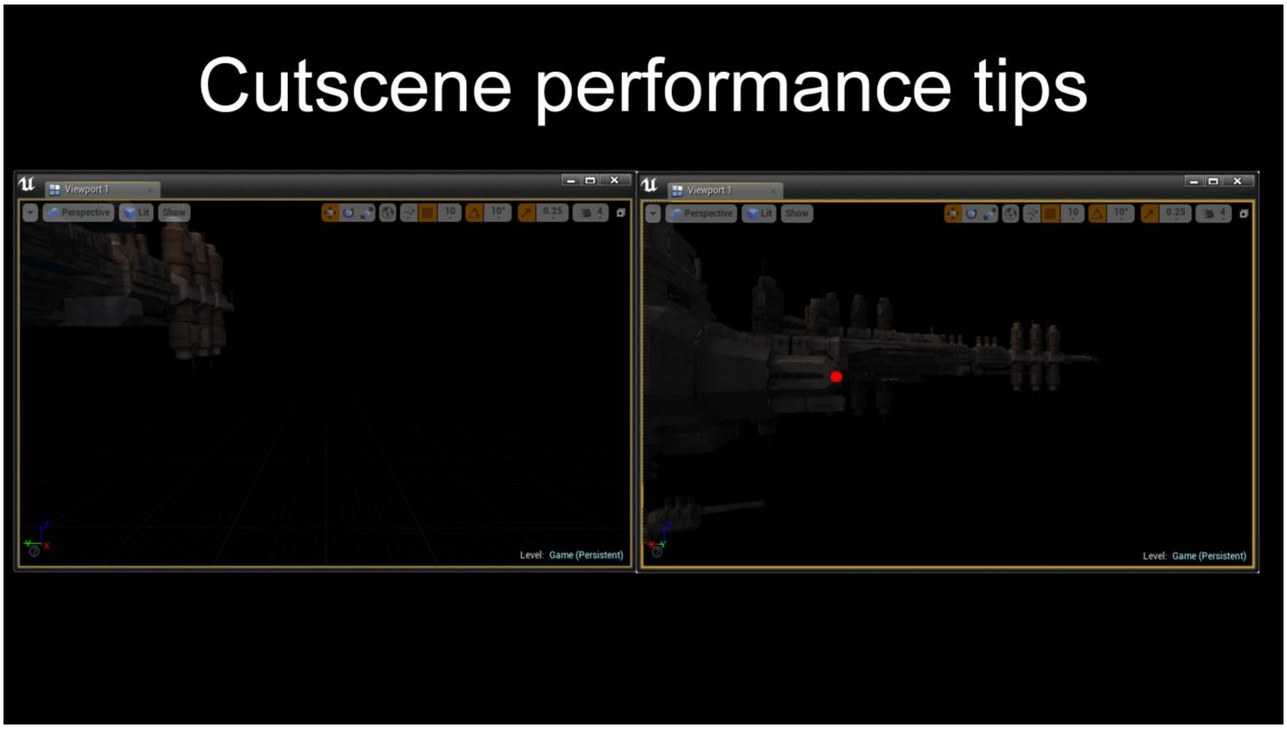
Matcap fake reflection

- Using Fully Rough for some object for higher performance
- But lose reflection
- Use Matcap reflection to Compensate

Allows reflective materials but with lower cost

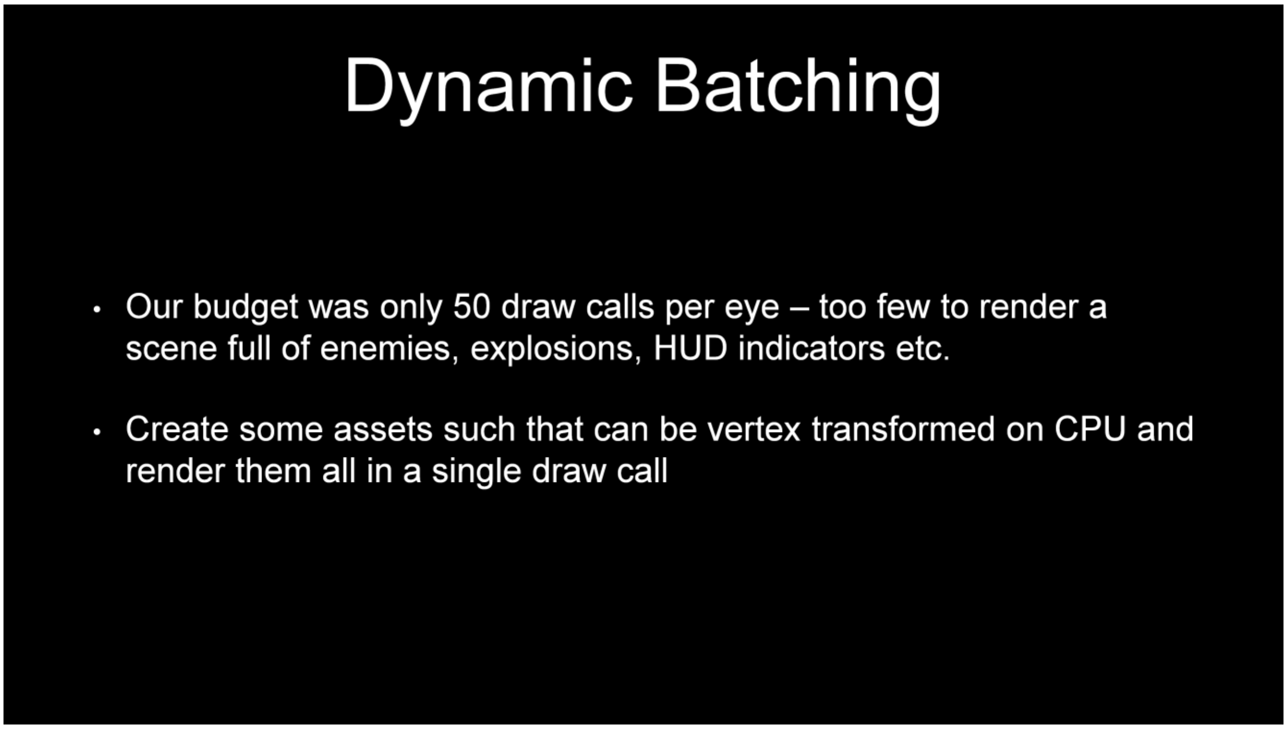
Cutscene performance tips

- Try to hide occluded object as much as possible
- Break big/complex mesh into pieces, use tricks to make sure player can only see part of pieces at a time
- Control visibility using matinee events
- Cumbersome work, but worth it



Cutscene performance tips

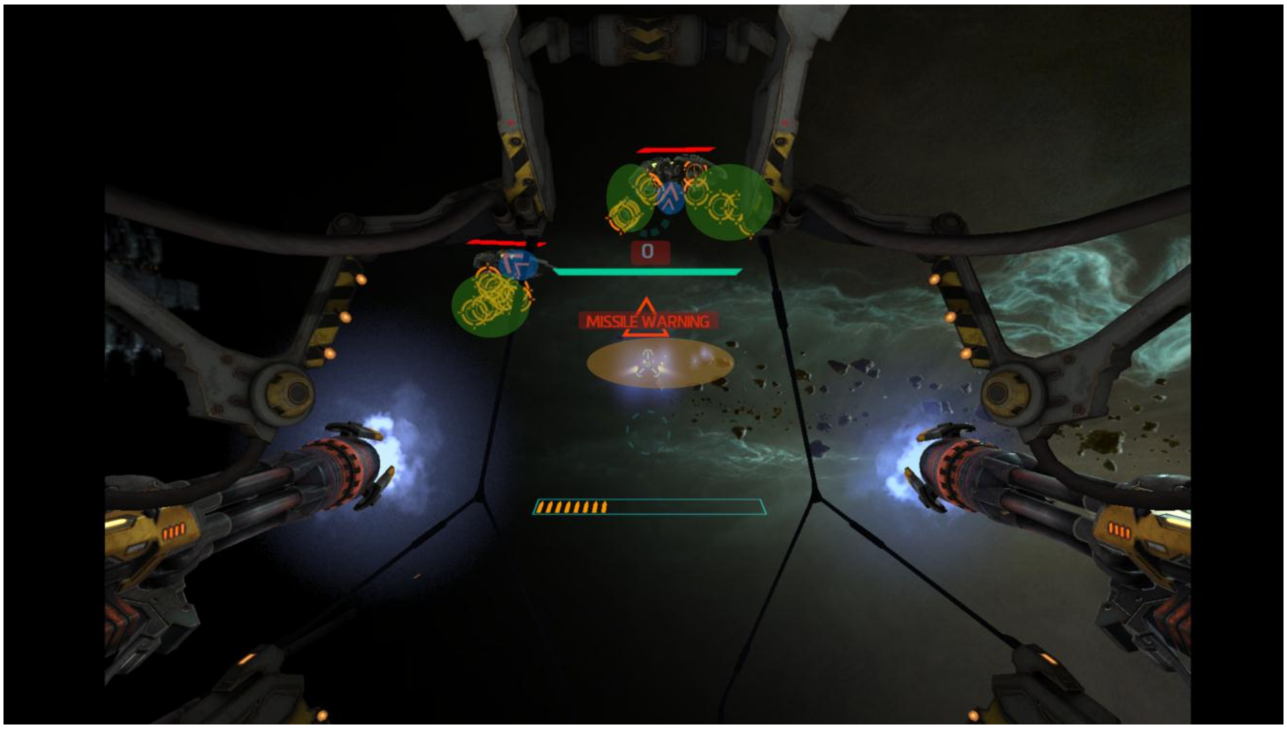
Required to keep quality as high as possible by simply turning off what can't be seen



Dynamic Batching

- Our budget was only 50 draw calls per eye – too few to render a scene full of enemies, explosions, HUD indicators etc.
- Create some assets such that can be vertex transformed on CPU and render them all in a single draw call

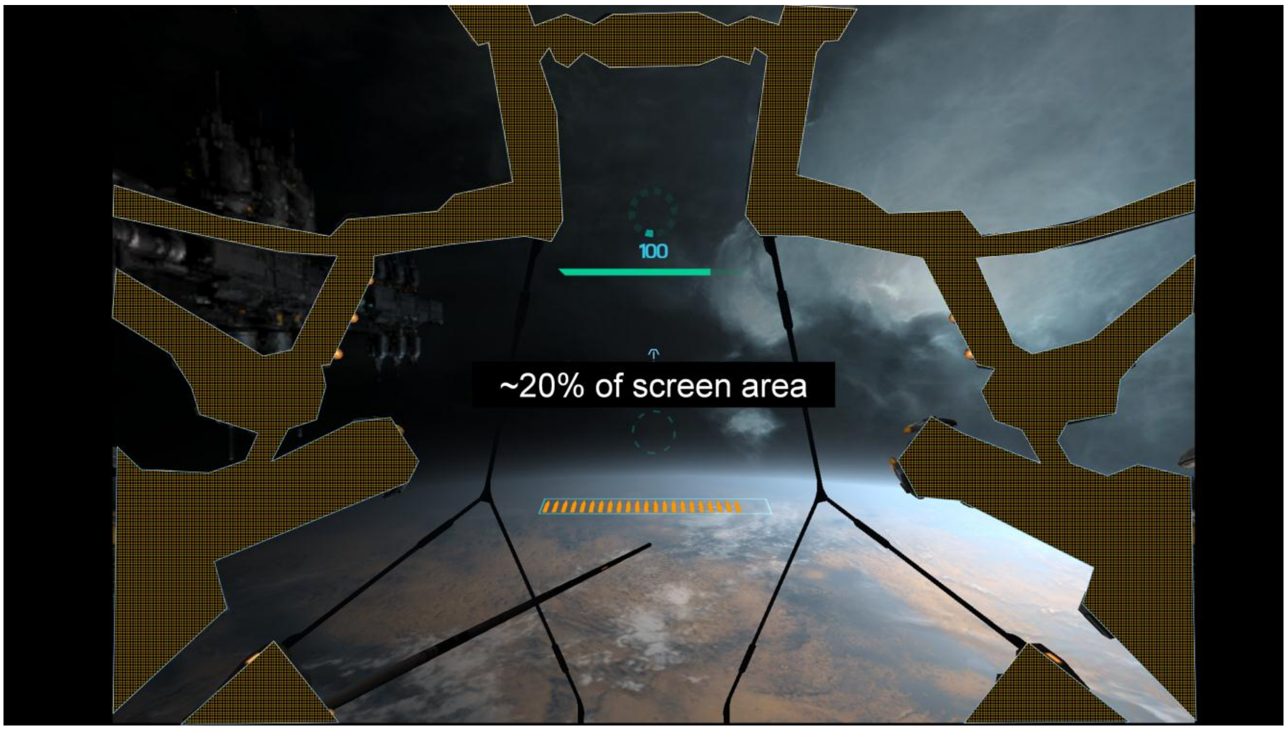
There was a per vertex CPU cost to this optimization – so it was important when creating assets that could be batched to have a reasonably low vertex count.



This screen shot of Gunjack showing several enemies firing missiles is a good example of this optimization working well – highlighted in each colour is each batch that were each rendered with a single draw call

Draw order

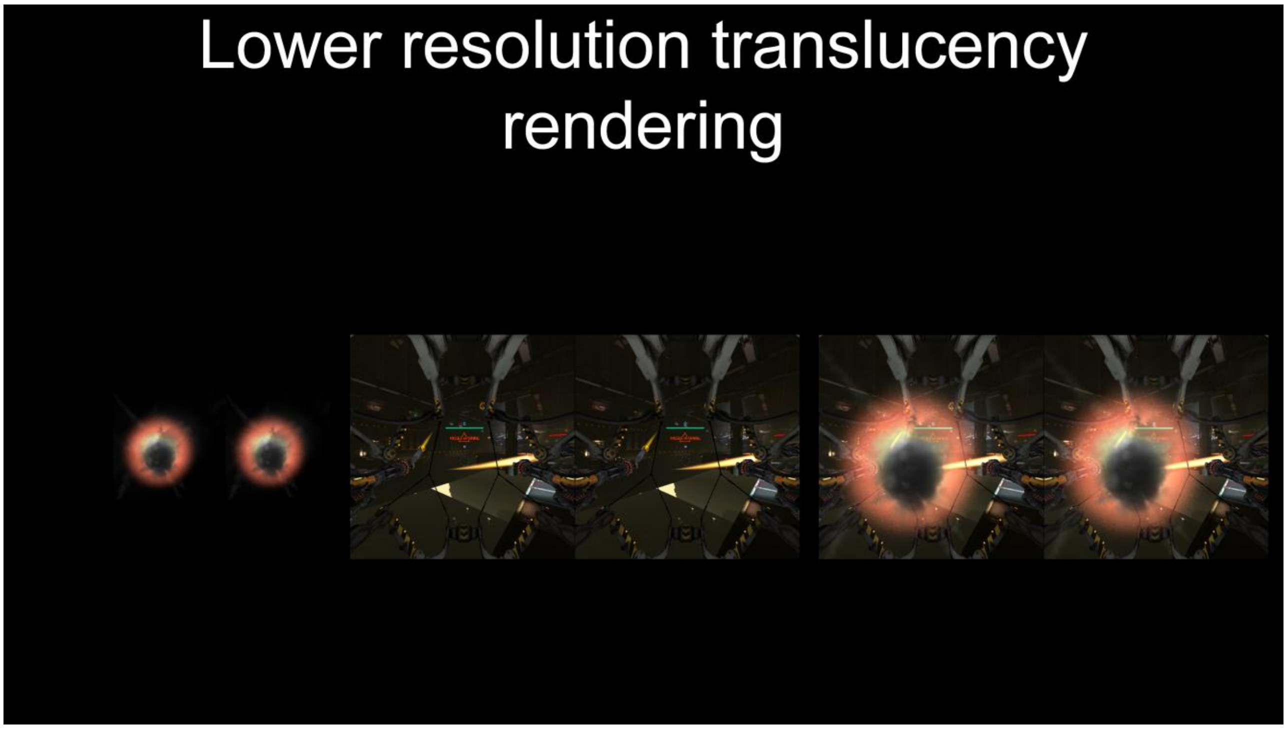
- Fill rate is also a precious resource
- If render near object first we can reject pixels behind it
- Lots of pixel fill saved by sort object from near to far in Gunjack



This screen shot shows a typical scene in Gunjack – the area highlighted actually takes up about 20% of screen area – so just ensuring it is rendered first reduces overdraw.

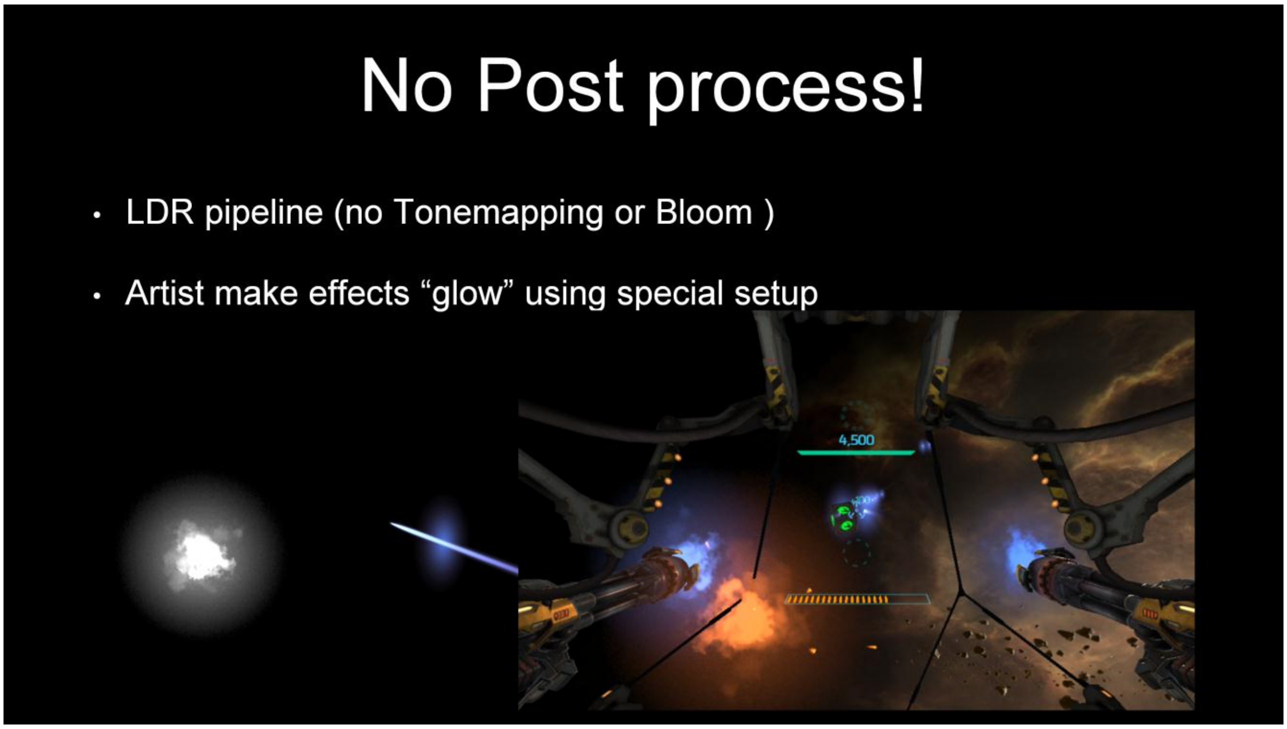
Lower resolution translucency rendering

- A method to reduce number translucent pixels rendered
- Artist mark some effects as “low res”
- Low res effects are rendered to a small rendertarget
- and composited to the full res screen in the custom post process shader



Lower resolution translucency rendering

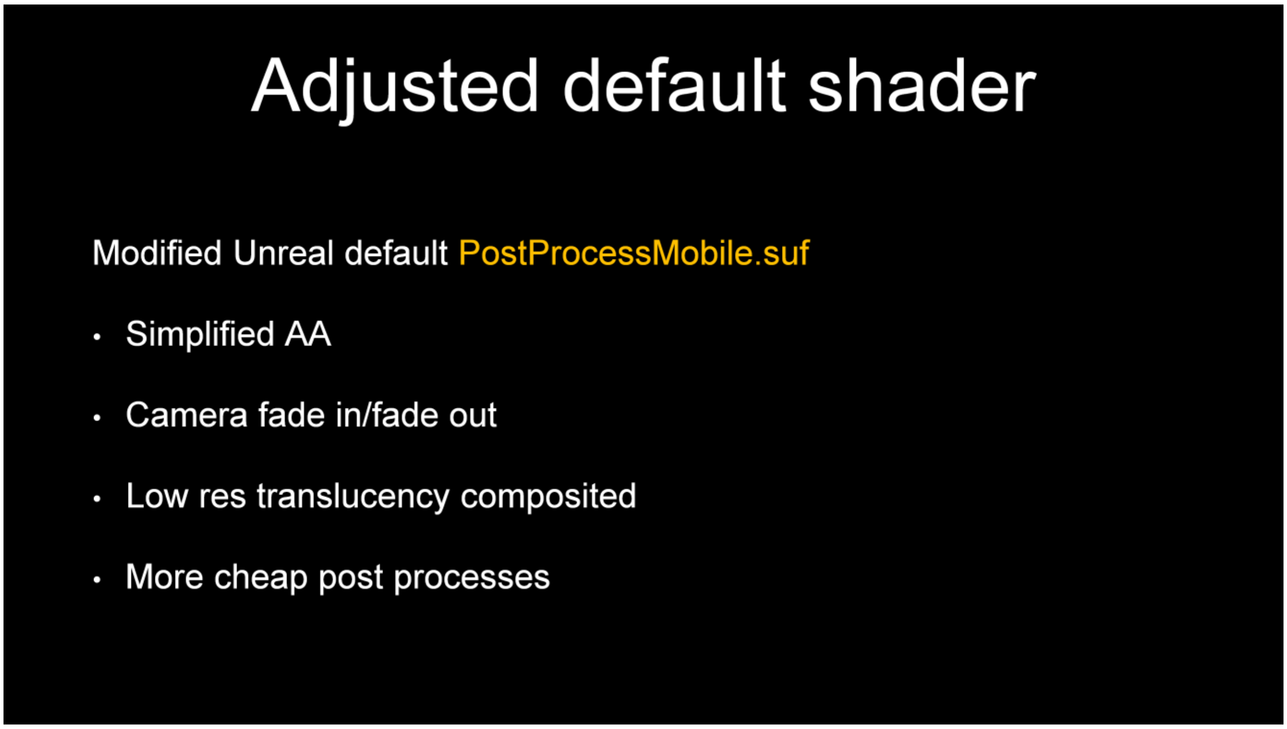
On the left is an explosion effect rendered at a lower resolution in the middle is a scene rendered at full resolution and on the right is the effect composited with he scene.



No Post process!

- LDR pipeline (no Tonemapping or Bloom)
- Artist make effects “glow” using special setup

Without HDR, effects artists still could use techniques from before HDR became possible – on the left you see effects rendered with extra glow sprites – which when composited into a scene on the right still looks very bright.



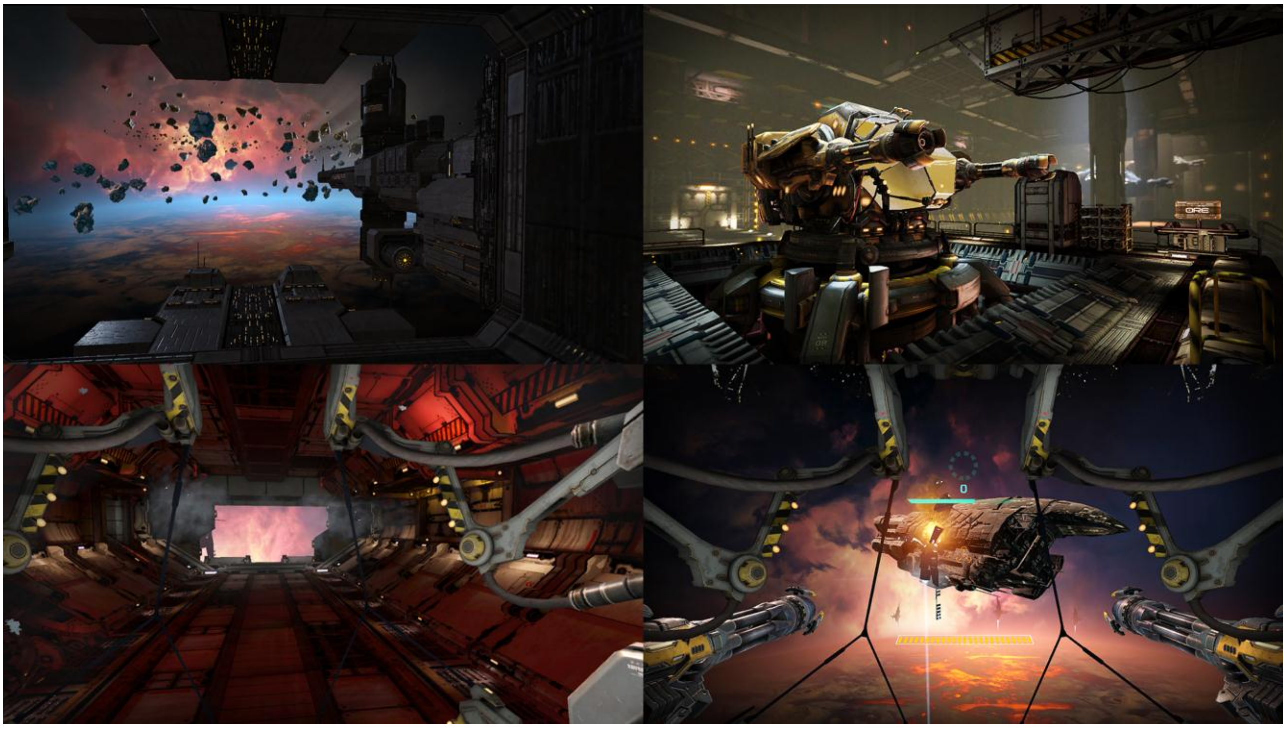
Adjusted default shader

Modified Unreal default PostProcessMobile.suf

- Simplified AA
- Camera fade in/fade out
- Low res translucency composited
- More cheap post processes

Topics Covered

- Graphics development approach
- Art content optimizations
- Rendering optimizations


ORE
0