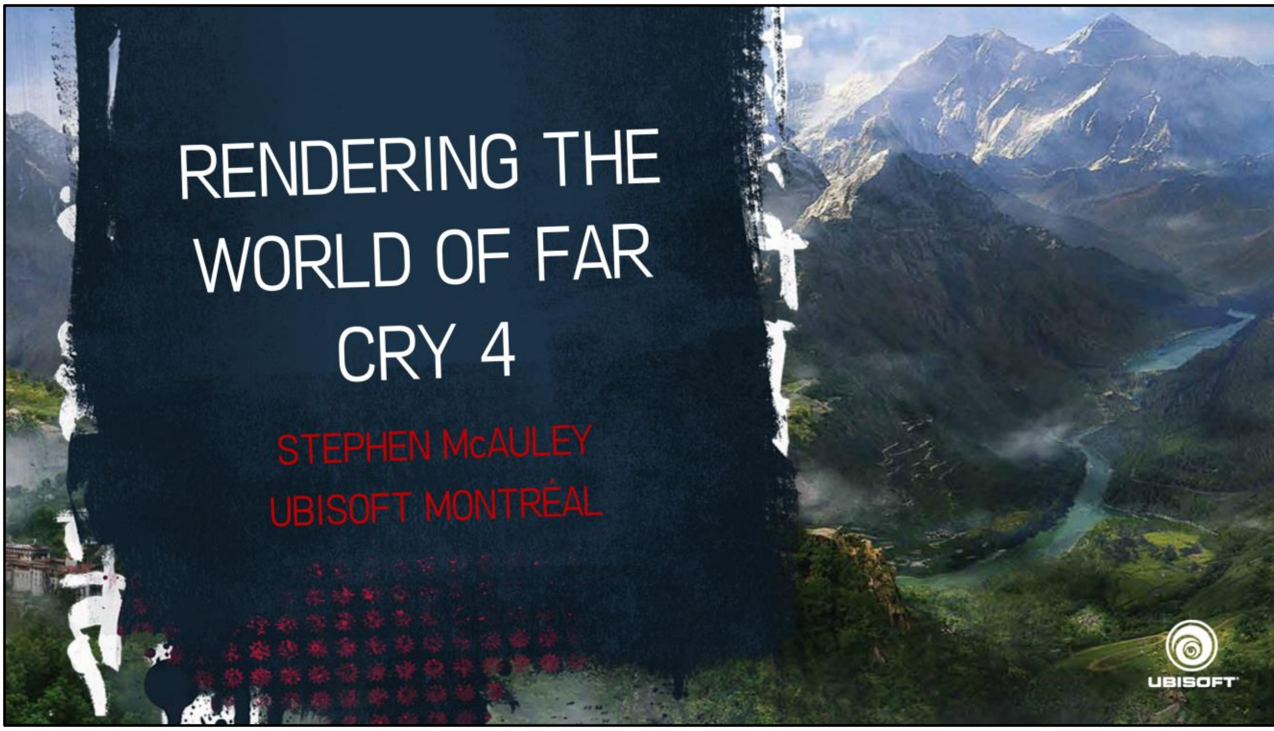
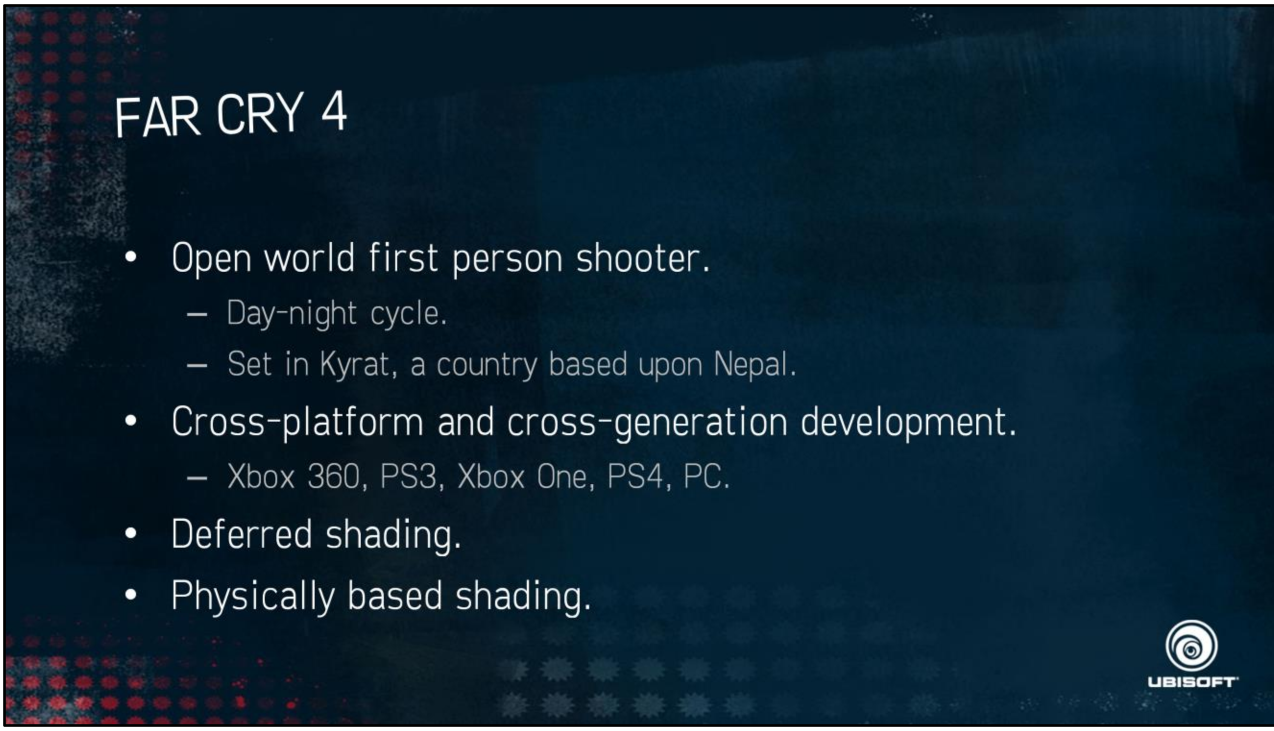


Let me preface my talk by saying that I'm the one lucky enough to be standing here today to talk about Far Cry 4, but really I'm presenting the work done by the whole graphics team at Ubisoft Montreal.

We started with the technology from Far Cry 3.

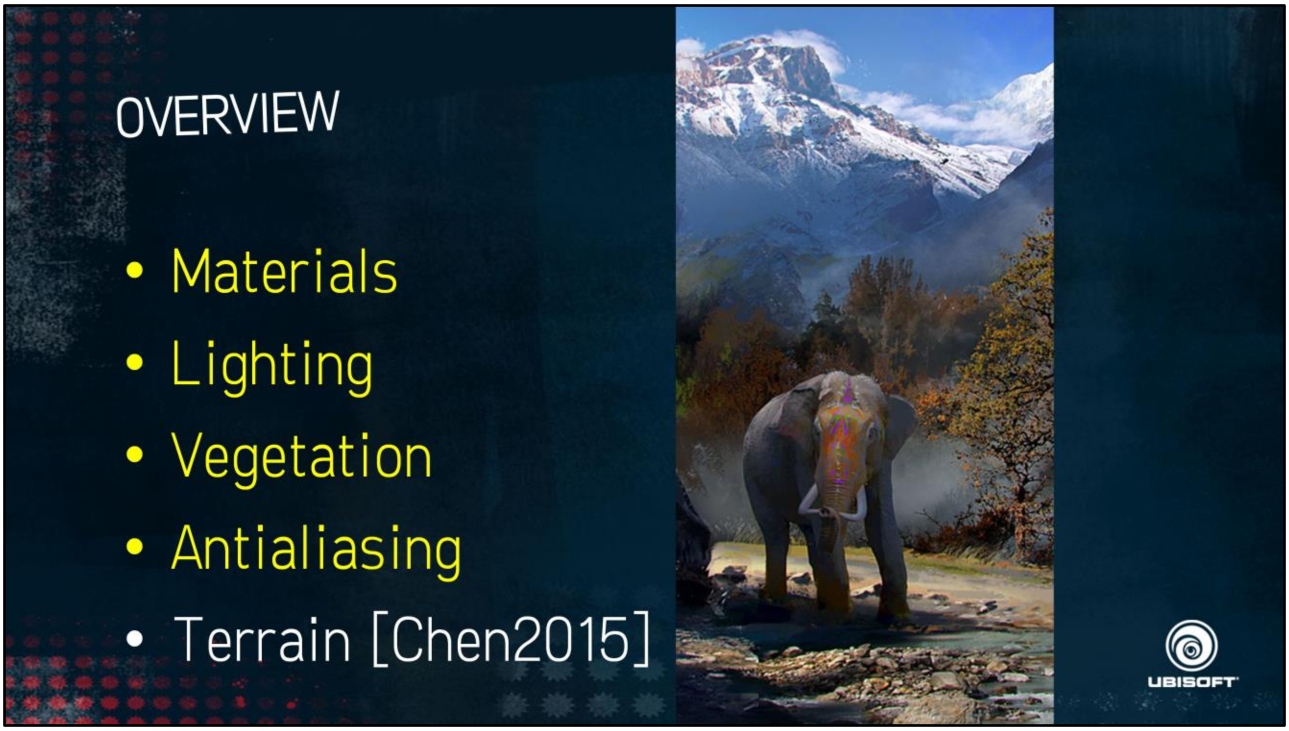
But we also knew where the problems were and what we wanted to improve from Far Cry 4. I'm going to go over some of these improvements.



With time of day changes, there is no hiding place…

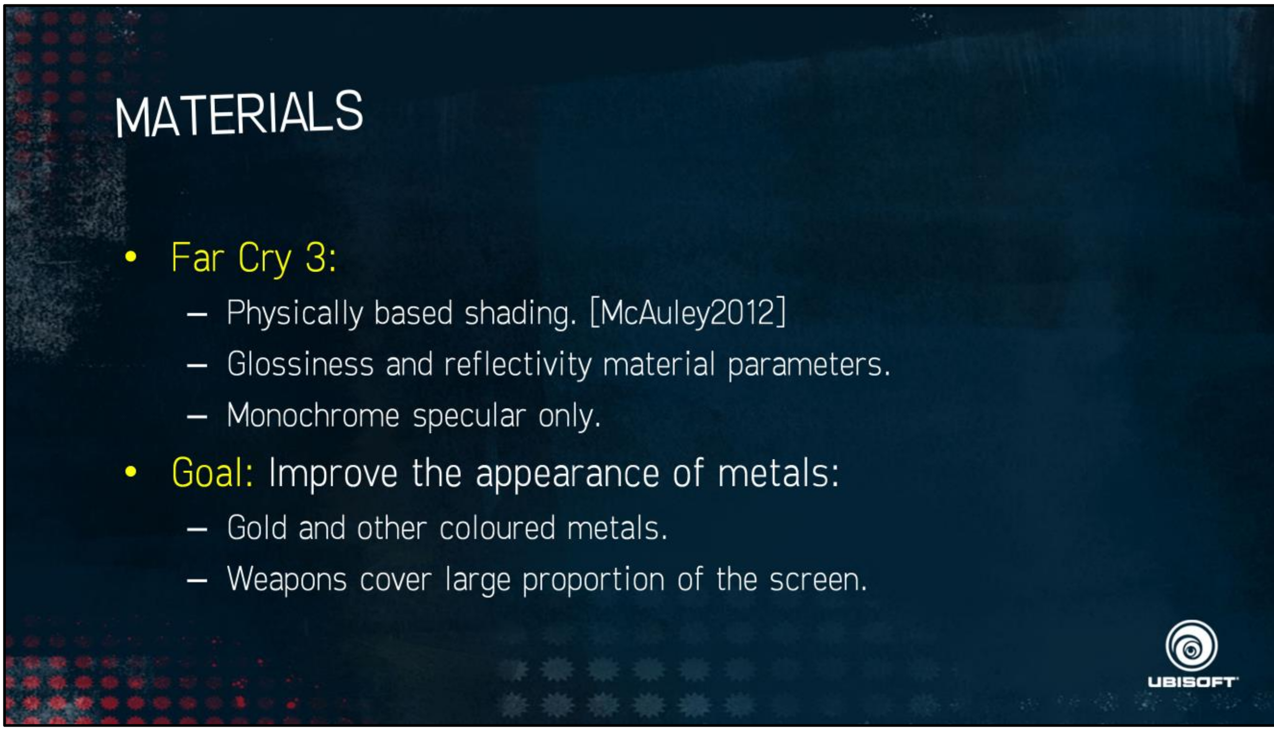
Our mandate as a graphics team was to lead on current-gen, keeping the last-gen engine the same as what shipped Far Cry 3. That gave us a lot of constraints as we had to keep the last-gen working, which affected a lot of our decisions throughout the project. By the end of the project, as was probably inevitable, we had to go back to the PS3 and Xbox 360 and polish those up, but it meant that we had a better product than FC3 on all platforms.

But thus today, I'm going to be talking almost exclusively about our work on Xbox One and PS4, but I'll drop a few titbits of information about the old consoles when I get the chance.



We focused on five main areas of improvement, but I'm only going to talk about the first four today. Hopefully you all attended Ka Chen's presentation earlier today, which talked about the virtual texturing we developed to improve our terrain rendering.


Rendering the World of Far Cry 4
MATERIALS

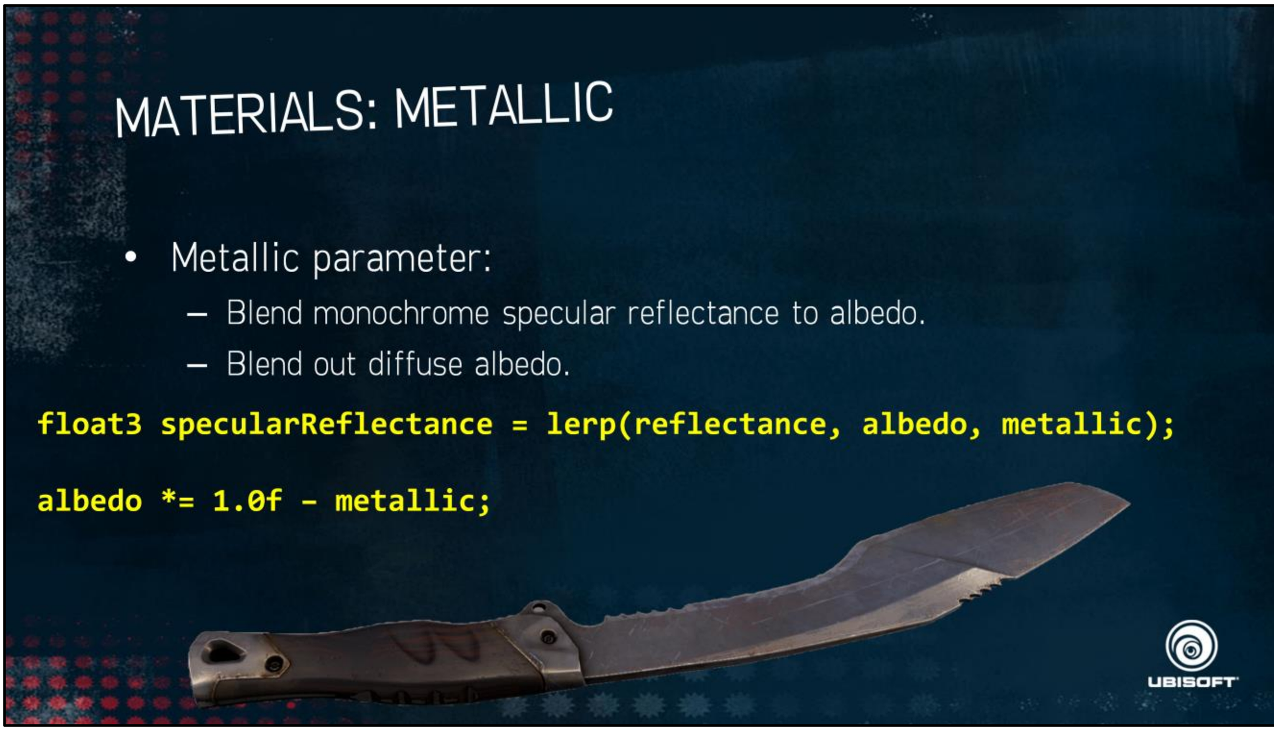


I spoke about physically based shading in Far Cry 3 at SIGGRAPH 2012.

Like everyone else, we're using the Disney BRDF. Disney call the "reflectance" parameter "specular", but I think the former works a lot better, plus they have "roughness" instead of "glossiness", but we had to stick with the latter for legacy reasons. We'll change it in the future.

We tried the Disney diffuse but it just didn't seem to make enough of a difference.

We’re going to talk about metallic and anisotropy today.

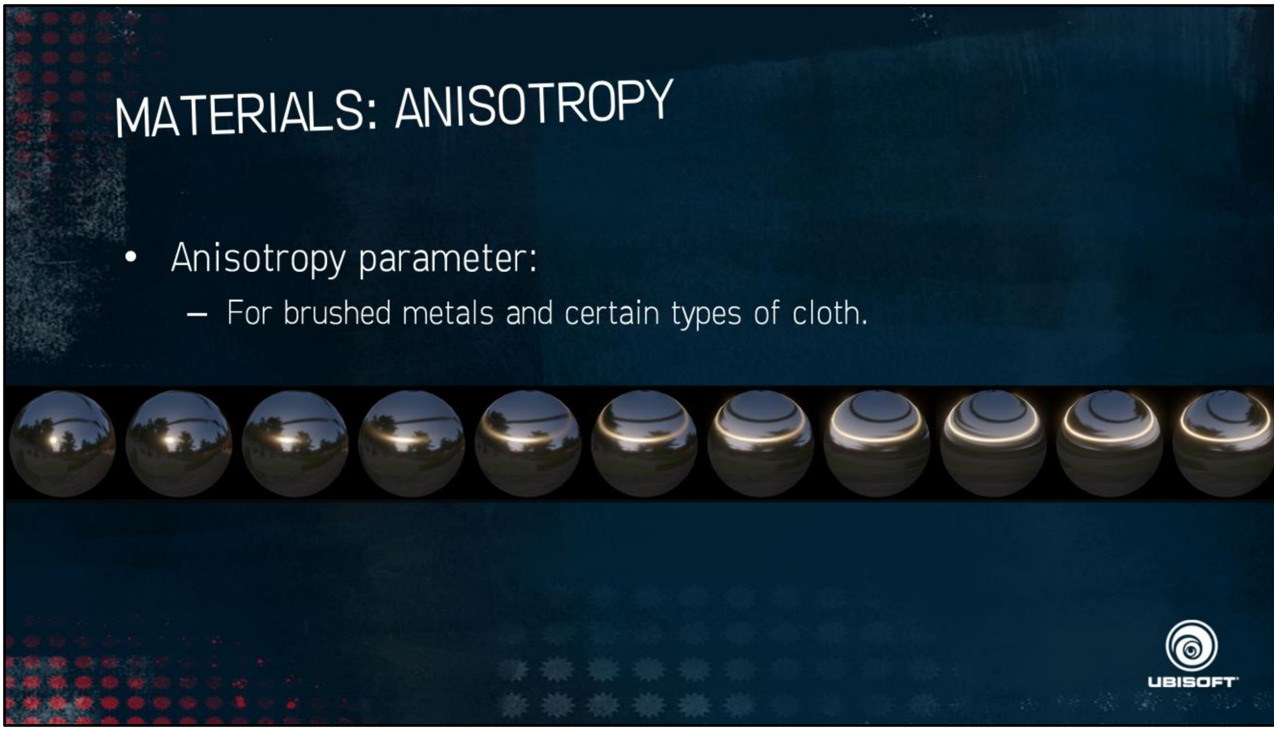


If we used a full specular colour, we'd have to find three G-Buffer channels which wasn't an option on PS3 and 360. Instead, we could just replace the reflectivity channel.

MATERIALS: METALLICITY

- Metallic parameter:
  - Single channel in G-Buffer for coloured specular.
  - Fix reflectance at 0.03 for last-gen.
  - Artists need training:
    - Behaviour of “diffuse” textures changes with metallic.
    - Metal values much brighter than for diffuse albedo.

UBISOFT



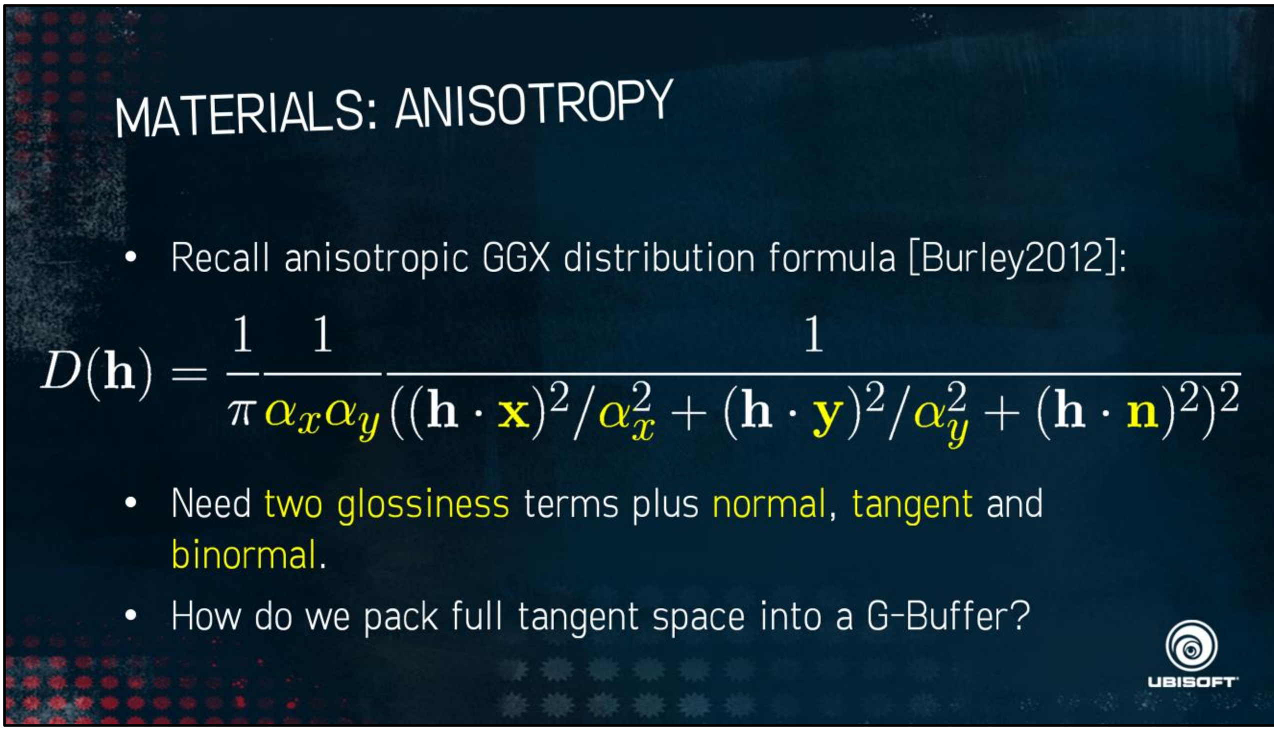
If we used a full specular colour, we'd have to find three G-Buffer channels which wasn't an option on PS3 and 360. Instead, we could just replace the reflectivity channel.

MATERIALS: ANISOTROPY

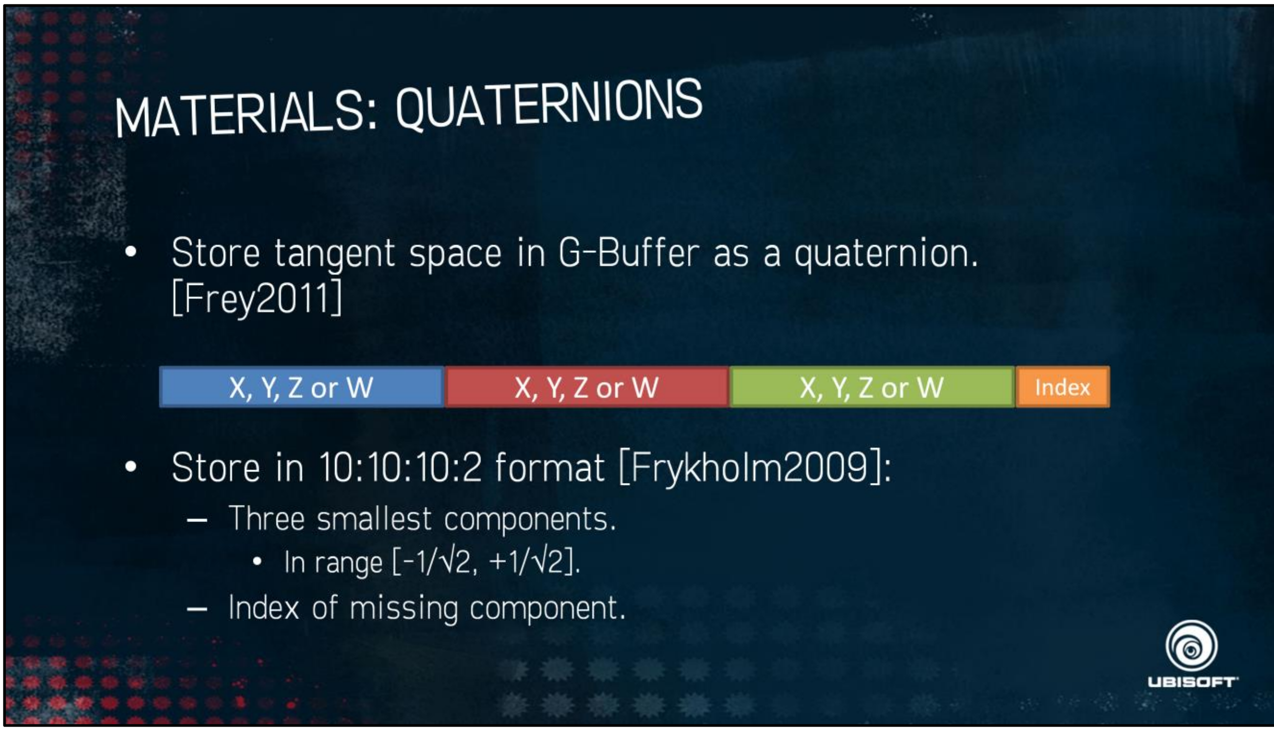
- Recall anisotropic GGX distribution formula [Burley2012]:

$$D(\mathbf{h}) = \frac{1}{\pi}\frac{1}{\alpha_x\alpha_y}\frac{1}{((\mathbf{h}\cdot\mathbf{x})^2/\alpha_x^2 + (\mathbf{h}\cdot\mathbf{y})^2/\alpha_y^2 + (\mathbf{h}\cdot\mathbf{n})^2)^2}$$

UBISOFT



I guess I should note hear that technically the alpha parameters are roughness, not glossiness, and we really should be calling glossiness "smoothness" as it's a more accurate representation of what it is. Still, we kept with the legacy naming from Far Cry 3.



To store tangent space compactly, we can actually steal ideas from animation systems. In particular, Crytek did a presentation on packing tangent space as quaternions for their animation system, as did Niklas Frykholm on the BitSquid blog.

The three smallest components are in the range [-1/√2, +1/√2] because if one was any bigger, it would have to be the biggest component. We can then rescale that range to [0, 1] for better precision.

MATERIALS: QUATERNIONS

- Find biggest component in only 5 instructions:
  - Use GCN-specific instructions. [Drobot2014a]

```
uint FindBiggestComponent(in float4 q)
{
    uint xyzIndex = CubeMapFaceID(q.x, q.y, q.z) * 0.5f;
    uint wIndex   = 3;

    bool wBiggest = abs(q.w) > max3(abs(q.x), abs(q.y), abs(q.z));

    return Select(wBiggest, wIndex, xyzIndex);
}
```

UBISOFT

For all the shader code for quaternion packing/unpacking, please see the appendix.

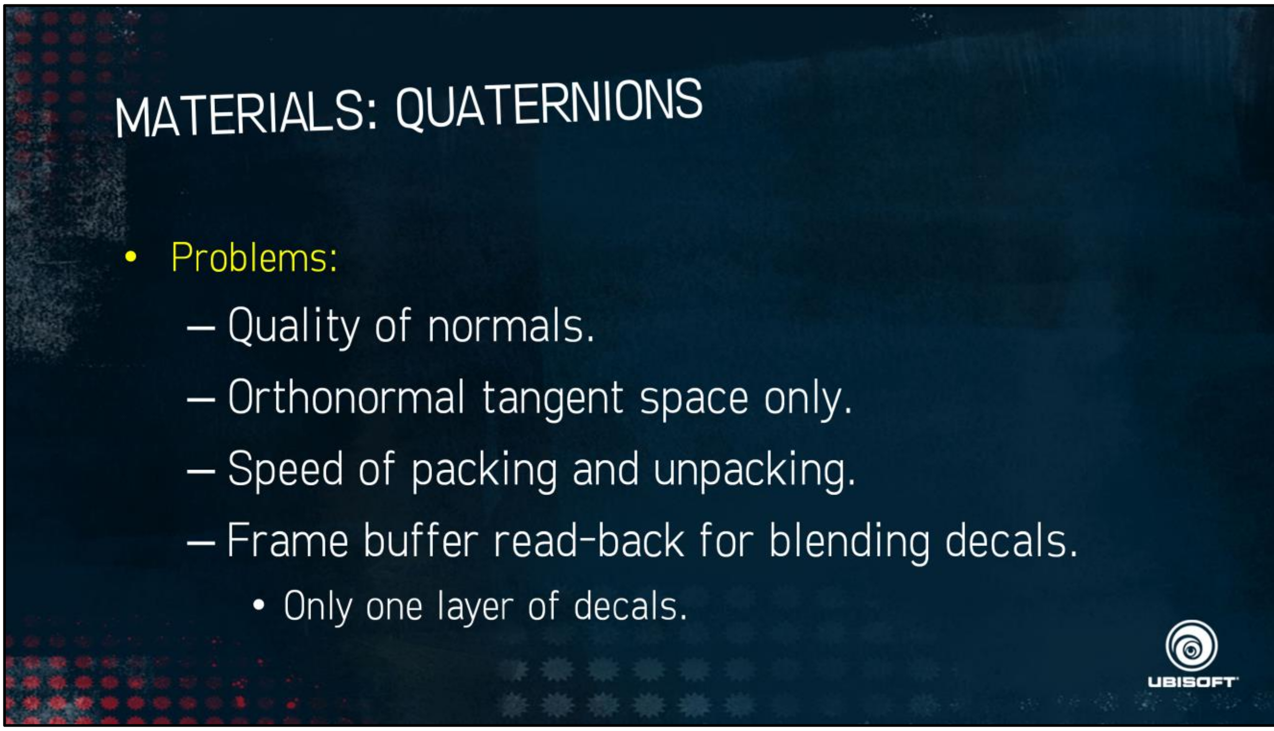
MATERIALS: QUATERNIONS

- Quality equivalent to naïve packing of normals in 8:8:8.
- Faceting on smooth surfaces:

Blinn-Phong
8:8:8 normals

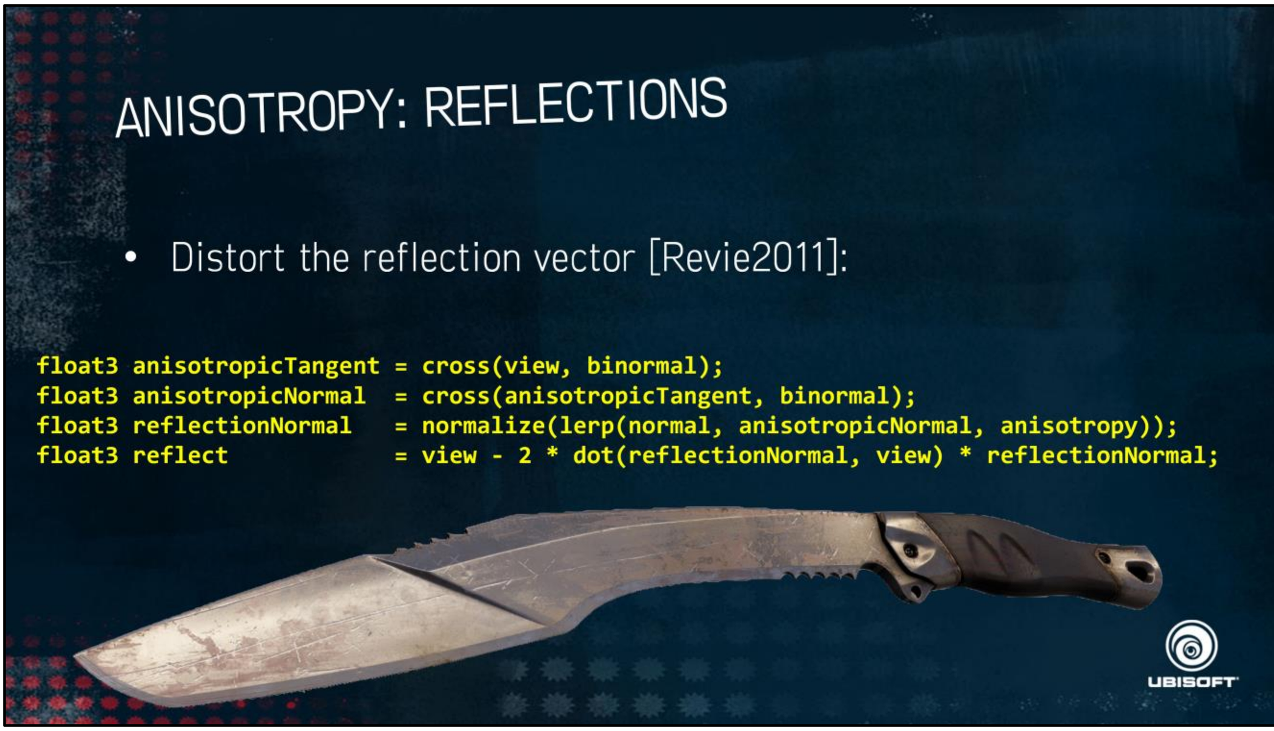
GGX
10:10:10:2 quaternions

UBISOFT



Orthonormal tangent space isn’t so bad, particularly because you can decouple the tangent space used for shading and the tangent space used for normals. (That’s a problem if you’re going to do LEAN mapping, but we weren’t.)

For blending decals, having virtual textures helps us a lot here, because all our terrain decals are baked into the virtual texture. For all other objects not using virtual texturing, such as decals on buildings, having only one layer was a limitation we presented to artists and they worked around.

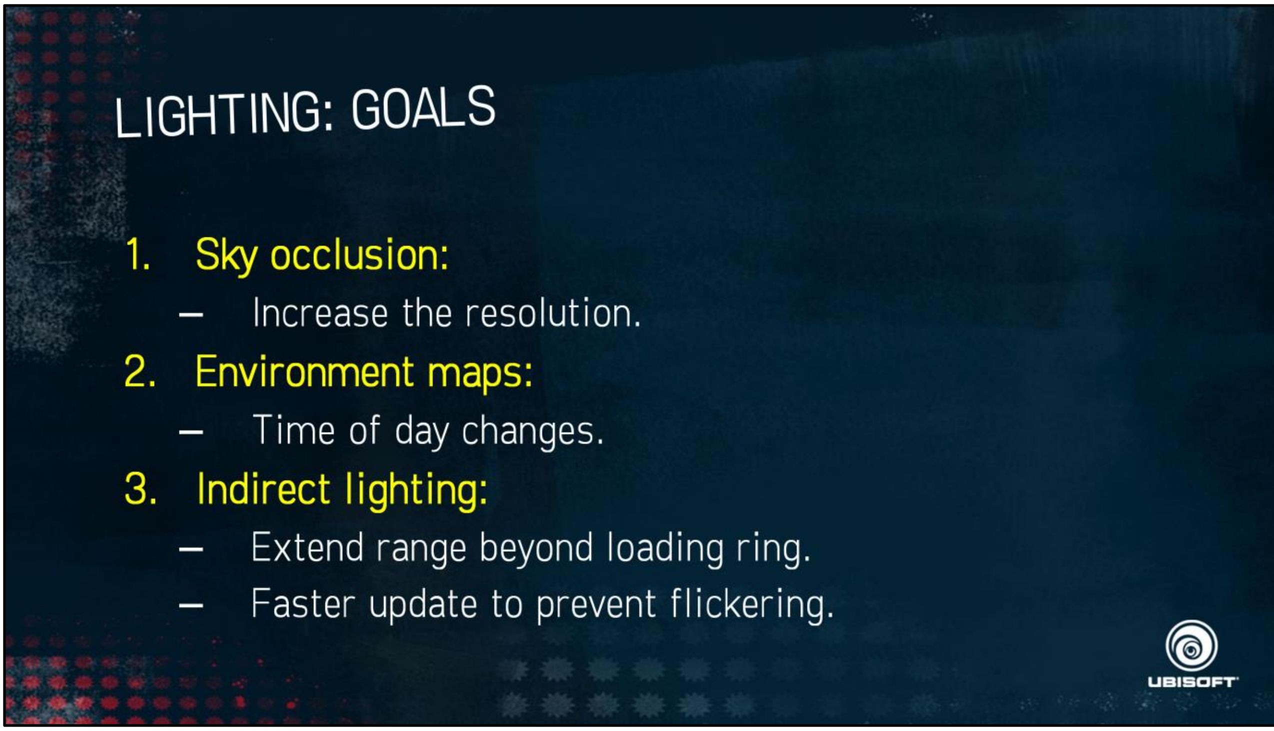


I'm grateful to Matt Pettineo for bringing Don Revie's article to my attention.

I'll be honest, it's far, far from perfect… but it's a hell of a lot faster than importance sampling and a hell of a lot better than doing nothing.

We solved an interesting problem with adding anisotropy… Our weapons team always complained that they wanted a fake specular term, as they didn't always see a highlight. With anisotropic specular, it completely stopped those complaints as they always felt they could see something cool going on from all angles. It's worth pointing out that this "fake specular" they wanted in fact therefore was a form of anisotropy – that's what they felt they were missing from the real world.


Rendering the World of Far Cry 4
LIGHTING



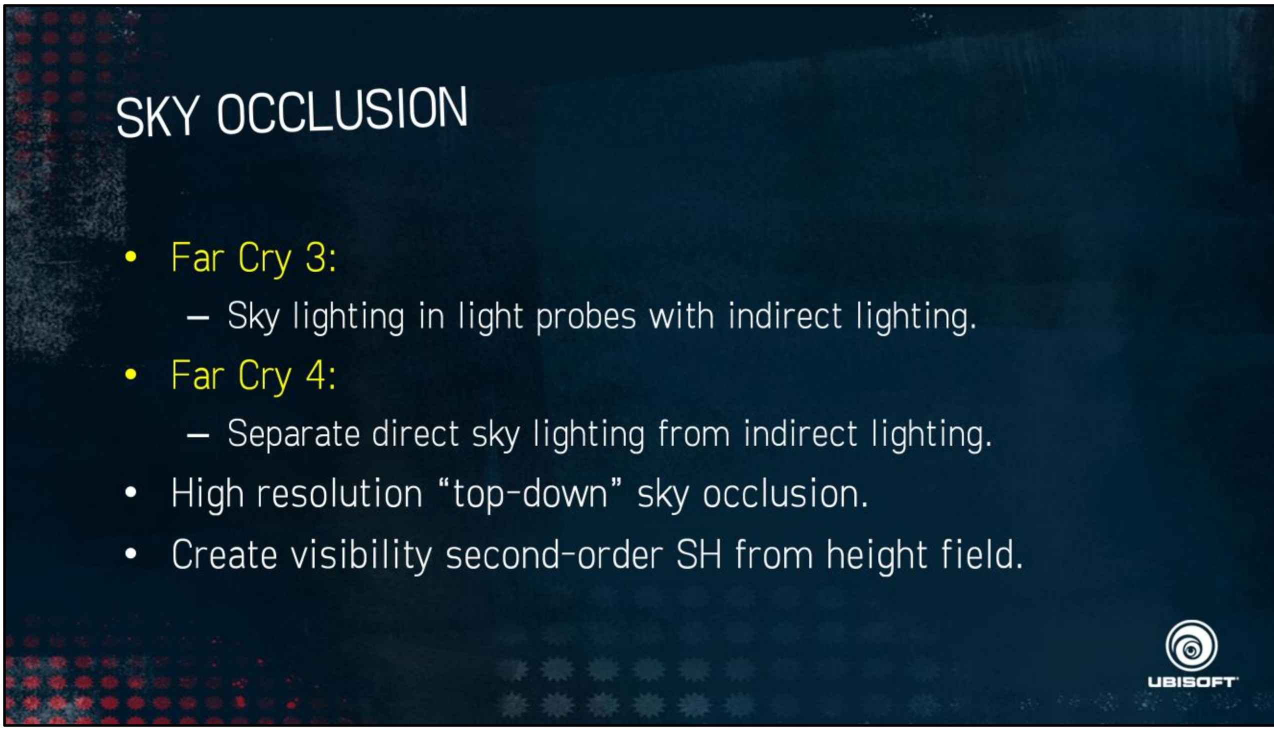
I’m mentioning work done by three people on our team. Jeremy Moore worked on the sky occlusion, and Gabriel Lassonde the environment maps, and myself on the indirect lighting.

Increasing the resolution of the sky occlusion was probably our priority over increasing the resolution of the indirect lighting, because it was less intrusive (important because of our cross-generation production) and it’s also easier – we knew it was achievable.

SKY LIGHTING

- Bruneton sky model and Preetham sun model.
- Generate lighting in third order SH.

UBISOFT



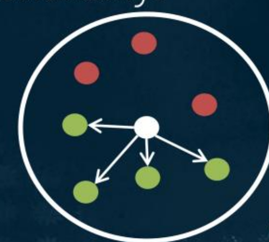
In other words, we generate a height map of our scene and use that to generate visibility information.

SKY OCCLUSION

- Height field rendered on demand:
  - World is split into 64x64m sectors.
  - Render whenever a sector becomes visible.
  - 0.281ms on GPU.
  - No streaming cost.
- Resolution of 25cm per texel.
- Blurred before generating SH visibility.

SKY OCCLUSION

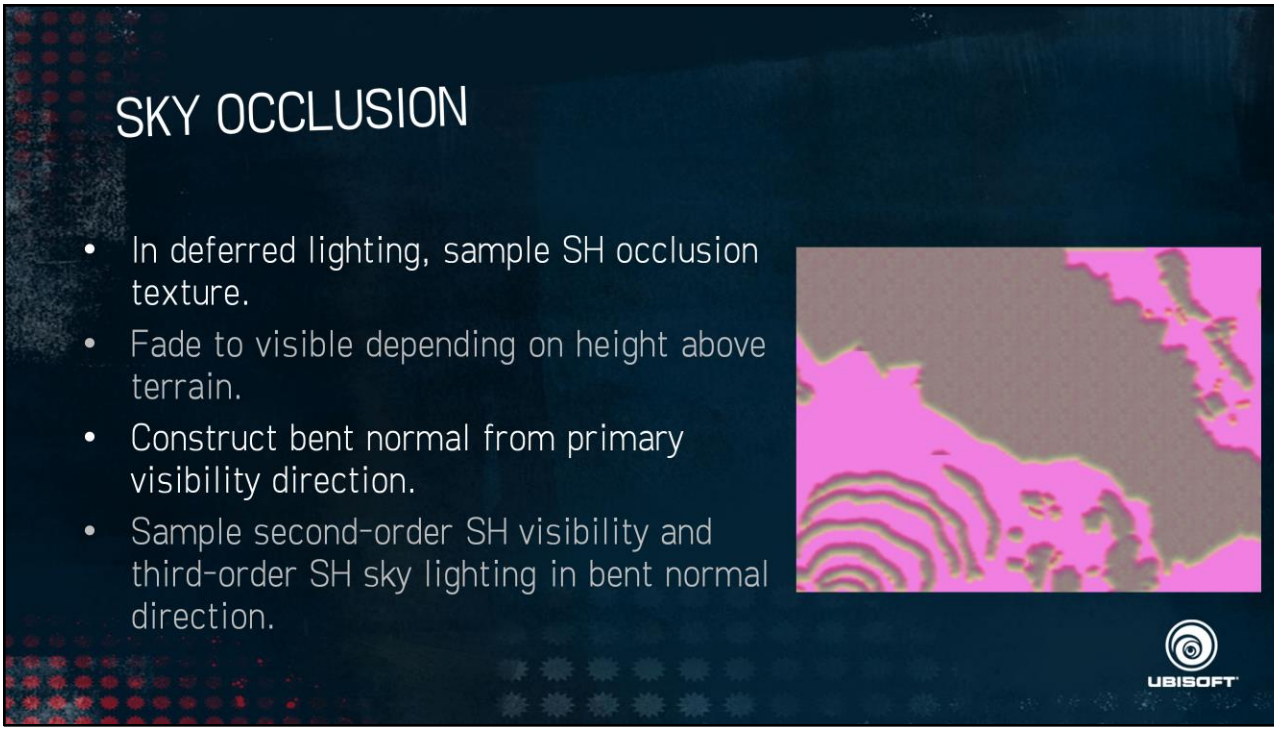
- SSAO-style approach.
- Sample blurred heightfield with rotated poisson disk.
- Store proportion of visible samples.
- Sum directions of visible samples for directionality.



SKY OCCLUSION

- Convert direction and visibility into SH.
- Generate SH for up normal and rotate:

```
float4 SH2SphericalCap(float cosA)
{
    float cos2A = 2 * cosA * cosA - 1;
    float4 sh = 0.0f;
    sh.x = g_shSphericalCapConstant0 * (1 - cosA);
    sh.z = g_shSphericalCapConstant2 * (1 - cos2A);
    return sh;
}
```



We fade the sky occlusion to fully visible based upon the height above the terrain – at the terrain height, we use the full sky occlusion value, but at the height stored in our blurred height map we count the pixel as fully visible.

LIGHTING: SKY OCCLUSION

- Bending the normal based on SH sky visibility:

```
float3 BentNormalFromSHVisibility(float3 normal, float4 shVis)
{
    float3 dir    = SH2GetPrimaryDirection(shVis);
    float  factor = saturate(1 - shVis.x / g_shSphericalCapConstant0);
    return normalize(normal + dir * factor);
}
```


WITHOUT SKY OCCLUSION

You might note the harsh falloff here – this was art directed, as we gave some controls to tweak the final result.

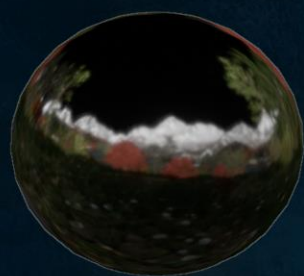
If you look at where the building meets the ground, you'll see the lighting now is a lot more consistent between the two surfaces.

How can a single cubemap have the right intensity at every time of day?

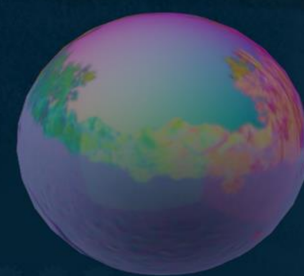

Why don't we relight the cubemap each frame?
UBISOFT

ENVIRONMENT MAP RELIGHTING

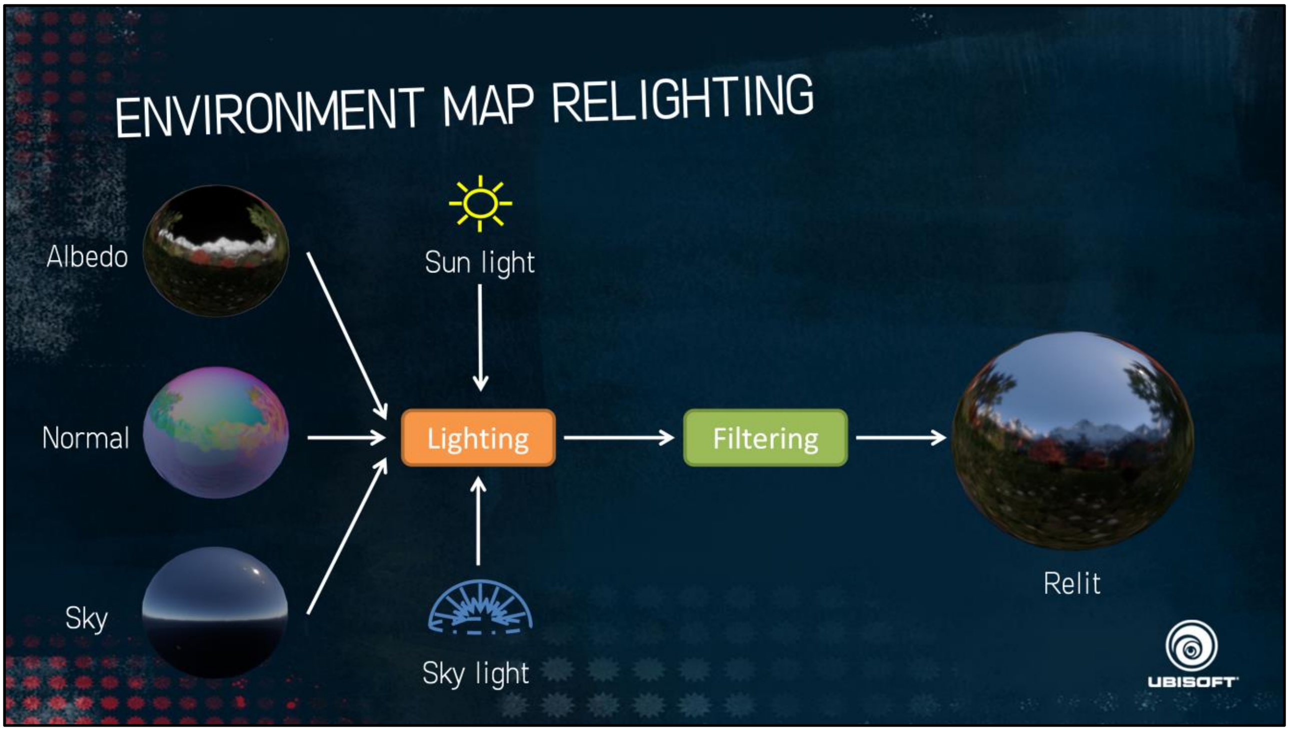
- G-Buffer cubemap:

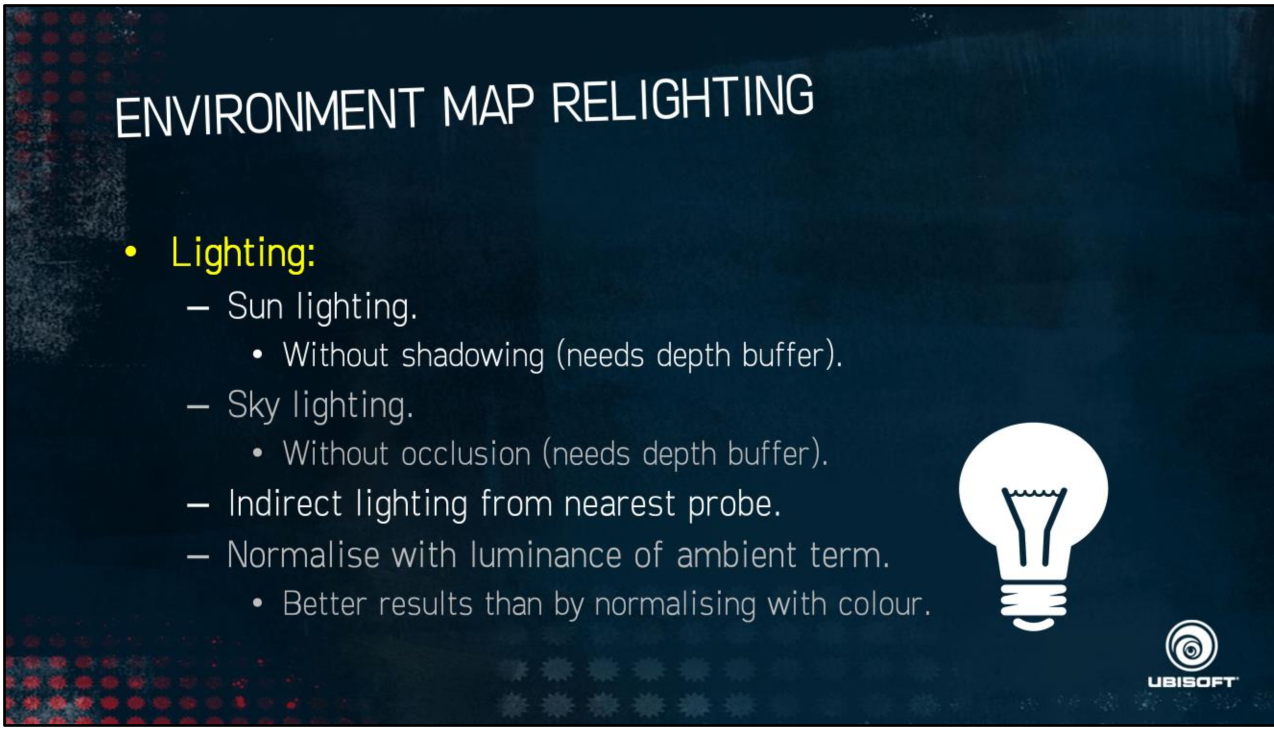


Albedo



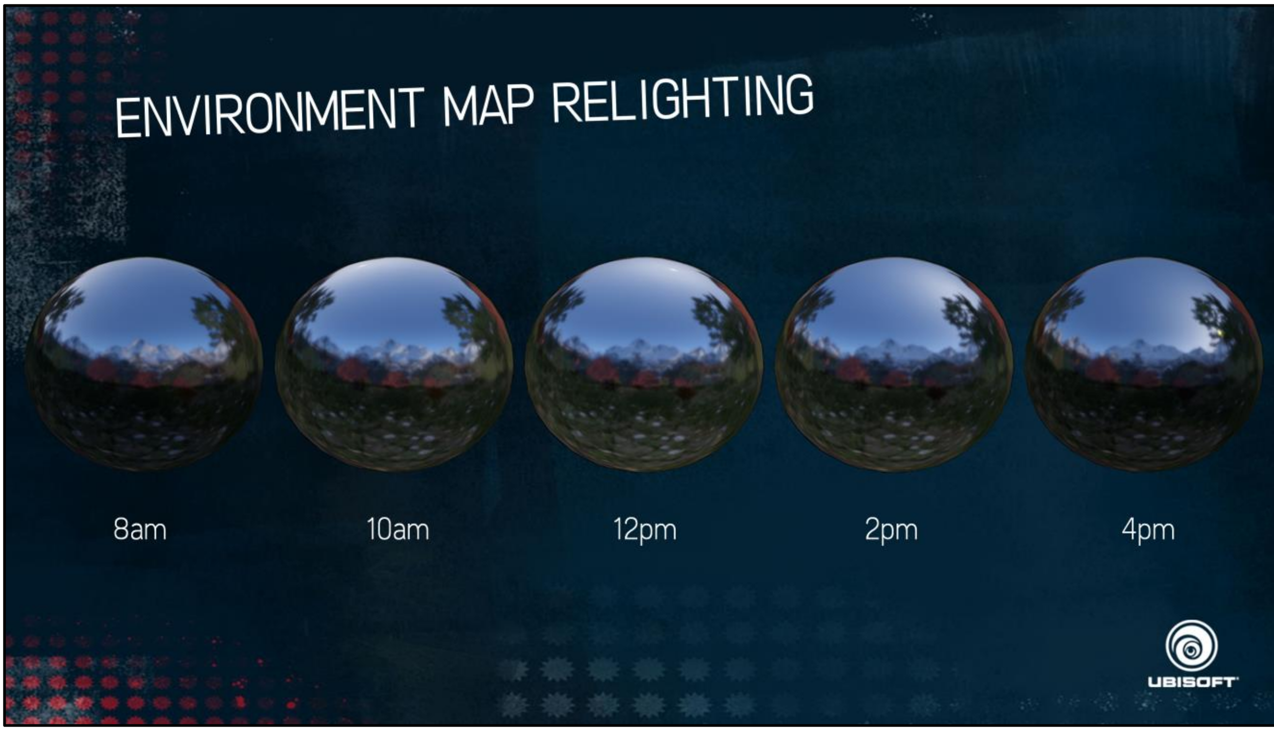
Normal


ENVIRONMENT MAP RELIGHTING
Albedo
Normal
Sky
Sun light
Lighting
Filtering
Sky light
Relit
UBISOFT



Shadowing and occlusion would require us to use the depth buffer to reconstruct position.

We use the luminance of the ambient term, rather than the colour, as we found that better preserved the correct result of the cube map at all times of day.



See the lighting of the mountains shifting.

We came up with the same method as Sebastien Lagarde and Charles de Rousiers for filtering environment maps at run time. Filtered importance sampling is absolutely key – so first, you generate mips with a simple box filter (which is fast), then you use filtered importance sampling to generate GGX filtered mips, which dramatically decreases the number of taps you need and the memory bandwidth.

Batching cube map faces (and if you have the ability, cube map mip levels) together is key for performance – otherwise you're running a lot of jobs on very small surfaces which has low GPU occupancy.

ENVIRONMENT MAP RELIGHTING

ENVIRONMENT MAP PERFORMANCE

- For a 128x128x6 R16G16B16A16F environment map:

Lighting	**0.066ms**
Mip Generation	**0.044ms**
Filtering	**0.275ms**

Timings taken from a PlayStation 4

- Bandwidth bound.

UBISOFT

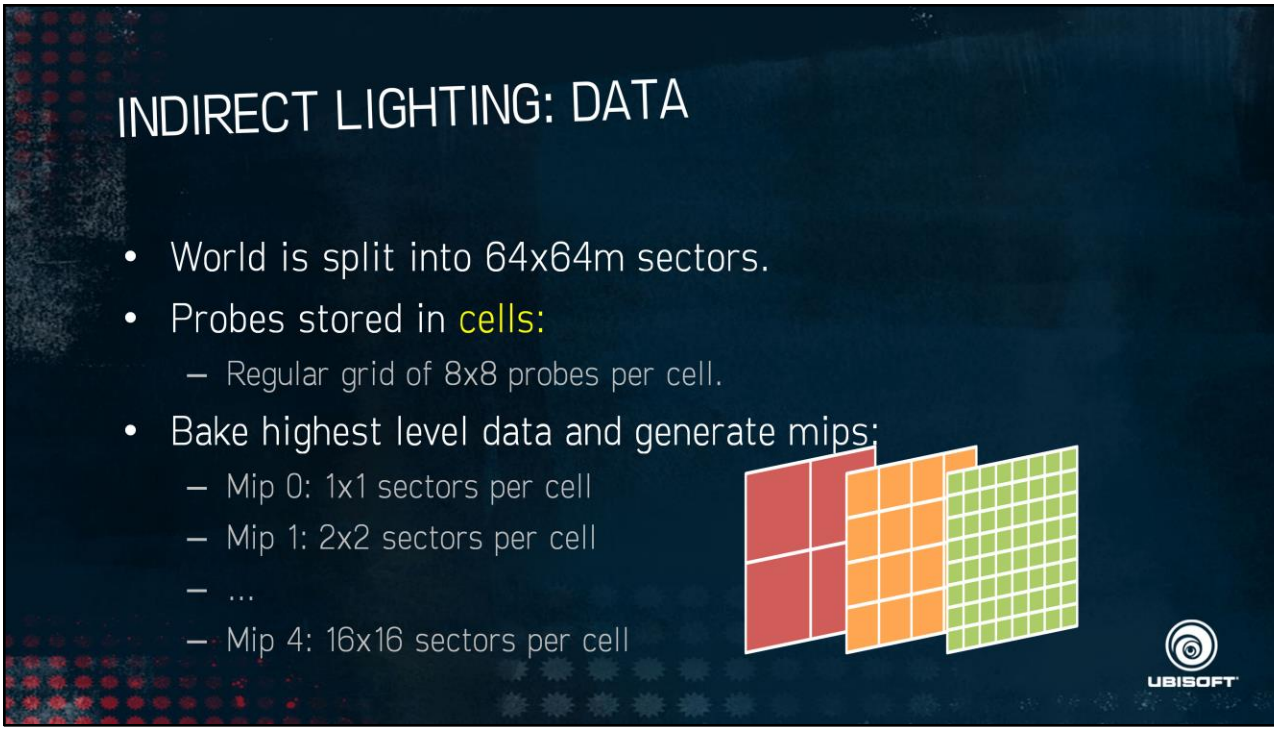
We're still bandwidth bound, despite filtered importance sampling. Our HDR texture format doesn't help here – David Cook at Microsoft has suggested we try R10G10B10A2 instead, but we haven't had time to experiment with that yet.

INDIRECT LIGHTING: OVERVIEW

- Far Cry 3:
  - Deferred radiance transfer volumes. [Stefanov2012]
  - Light probes that store radiance transfer information.
- Goals:
  - Use same light probe set for last- and current-gen.
  - Extend range of indirect lighting.
  - Faster updates by moving CPU work to GPU.

INDIRECT LIGHTING: OVERVIEW

- Offline:
  - Bake probes.
  - Radiance transfer information in second order SH.
- CPU:
  - Stream probe data.
  - Upload to GPU and update page table.
- GPU:
  - Calculate radiance transfer.
  - Inject probes into clip map and sample in deferred lighting.

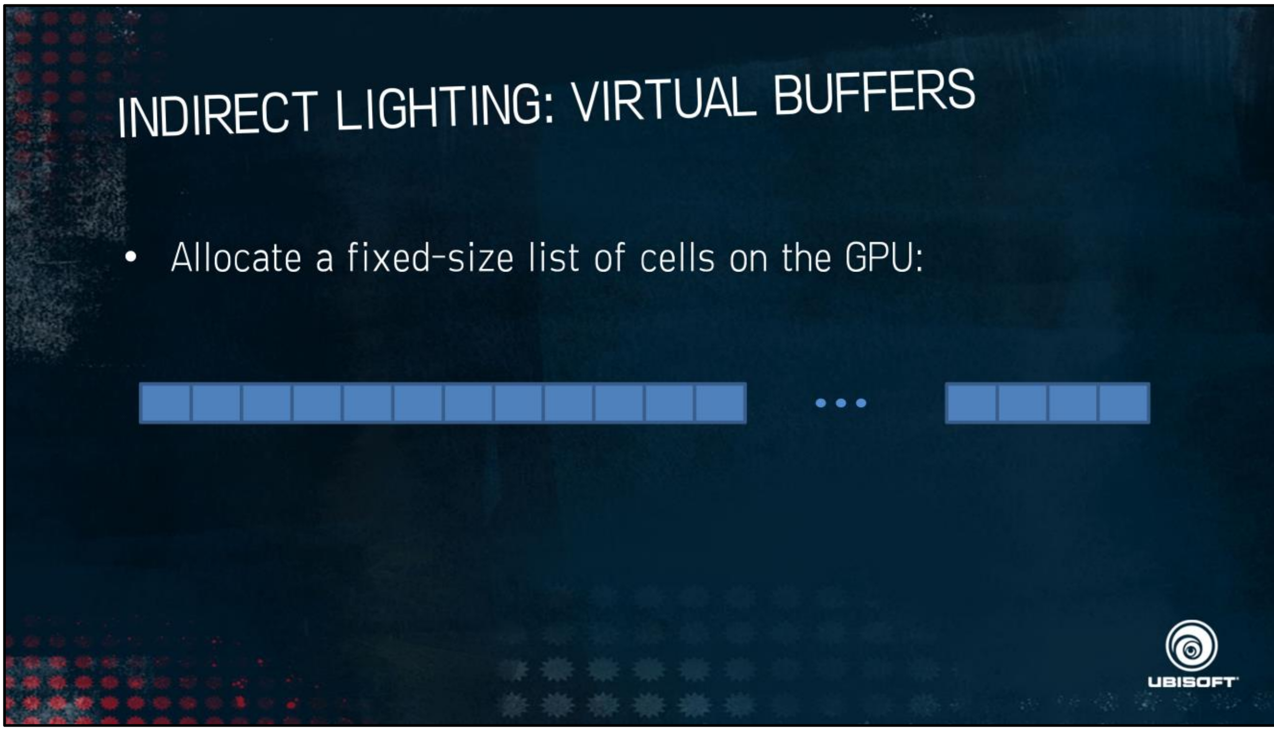


So a cell could be from any mip level and any size. In the end, we don't really care – we have a cell of light probes and we need to get a light probe within it.

INDIRECT LIGHTING: DATA

- Cell data:
  - Radiance transfer probe list:
    - 64 probes in an 8x8 regular grid.
  - SH probe list:
    - Calculated from radiance transfer probes once per frame
  - Dimensions.
    - Used for calculating index of probe to sample.

UBISOFT



How do we store our cells on the GPU? Well, we just allocate a fixed number of cells that we can have loaded at one time.

When I talk about virtual buffers here, I should probably mention this is all software – we`re not doing anything in hardware.

INDIRECT LIGHTING: VIRTUAL BUFFERS

- Allocate a fixed number of cells for each mip:

INDIRECT LIGHTING: VIRTUAL BUFFERS

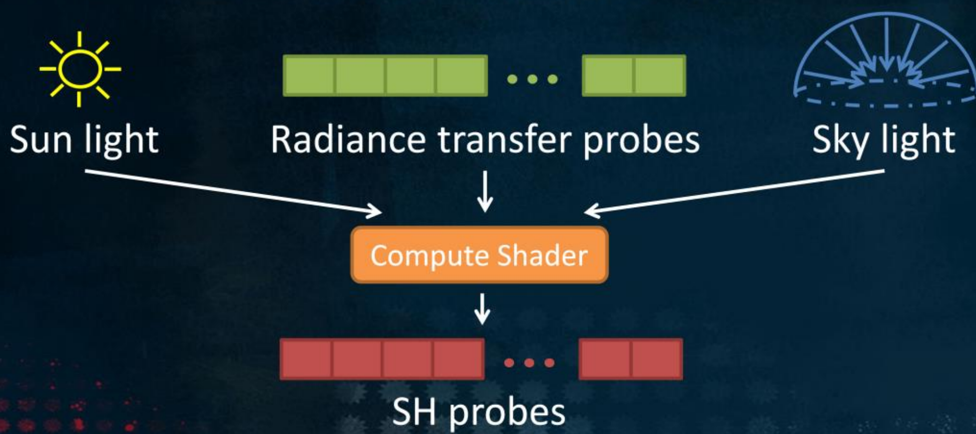
- As a cell is streamed in on CPU:
  - If a free cell is available on GPU, allocate.
  - If no cell is available, discard a more distant cell.
- As a cell is streamed out on CPU:
  - Mark cell as free on GPU.

UBISOFT

When I say allocate, obviously the memory is already allocated – we're just marking it as allocated and copying the data across from the CPU.

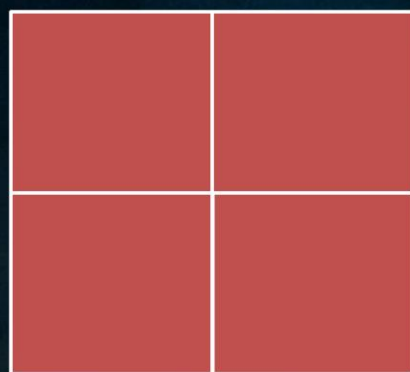
INDIRECT LIGHTING: RADIANCE TRANSFER

- Perform radiance transfer on entire cell list:



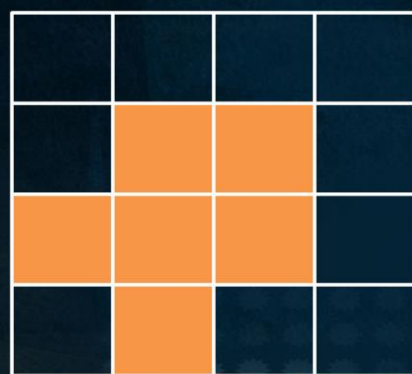
INDIRECT LIGHTING: PAGE TABLE

- Each mip level has a page table:



INDIRECT LIGHTING: PAGE TABLE

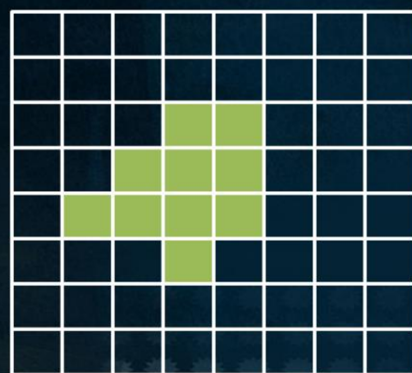
- Spatially indexed in a 2D grid in the XY plane:

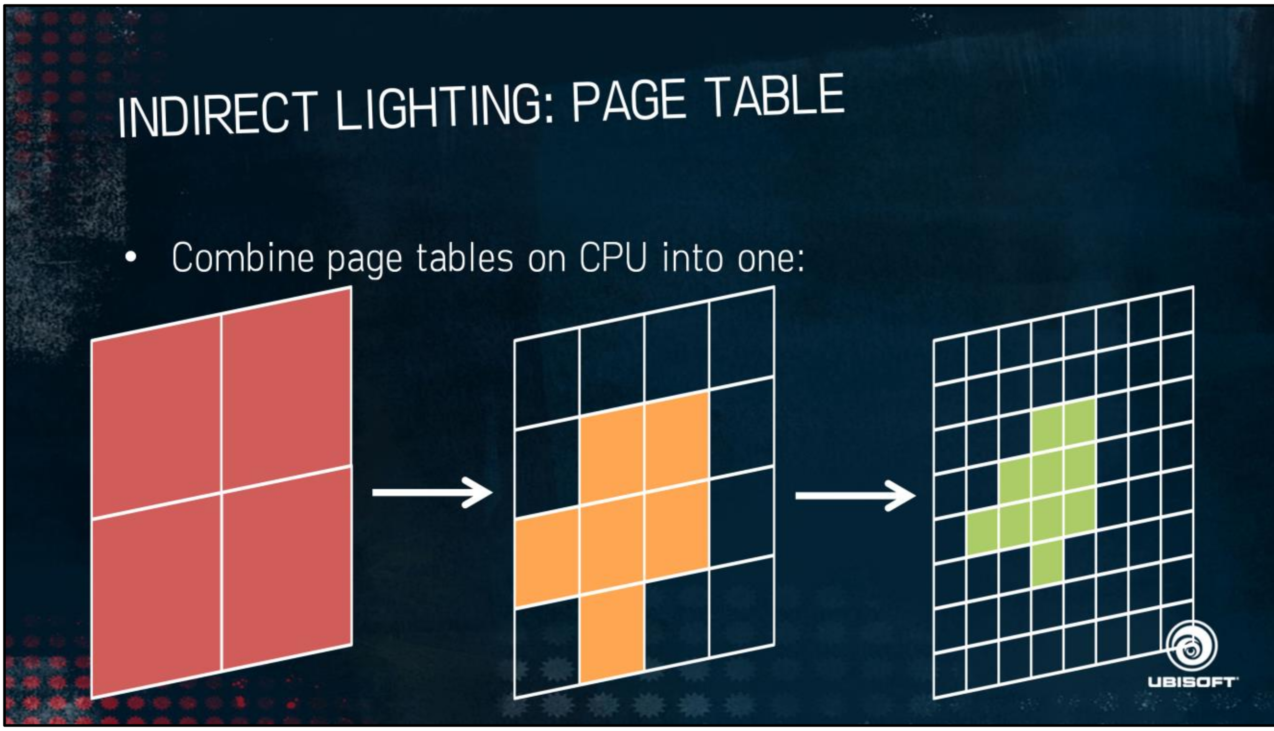


UBISOFT

INDIRECT LIGHTING: PAGE TABLE

- The page tables cover the entire world:

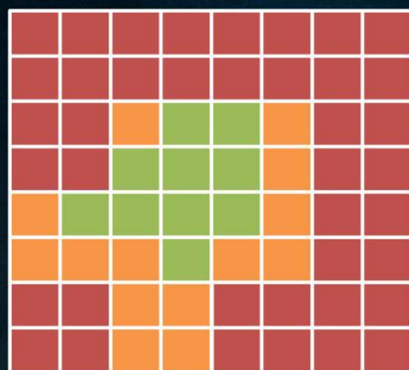




Honestly, my code for this is pretty slow. I might put it on GPU in the end.

INDIRECT LIGHTING: PAGE TABLE

- Combine page tables on CPU into one:

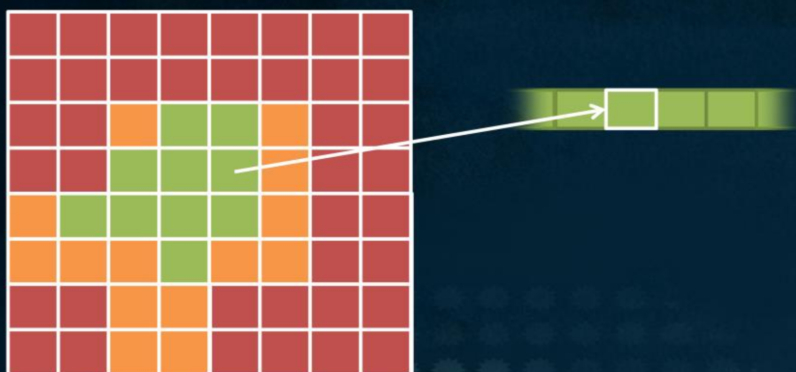


INDIRECT LIGHTING: PAGE TABLE

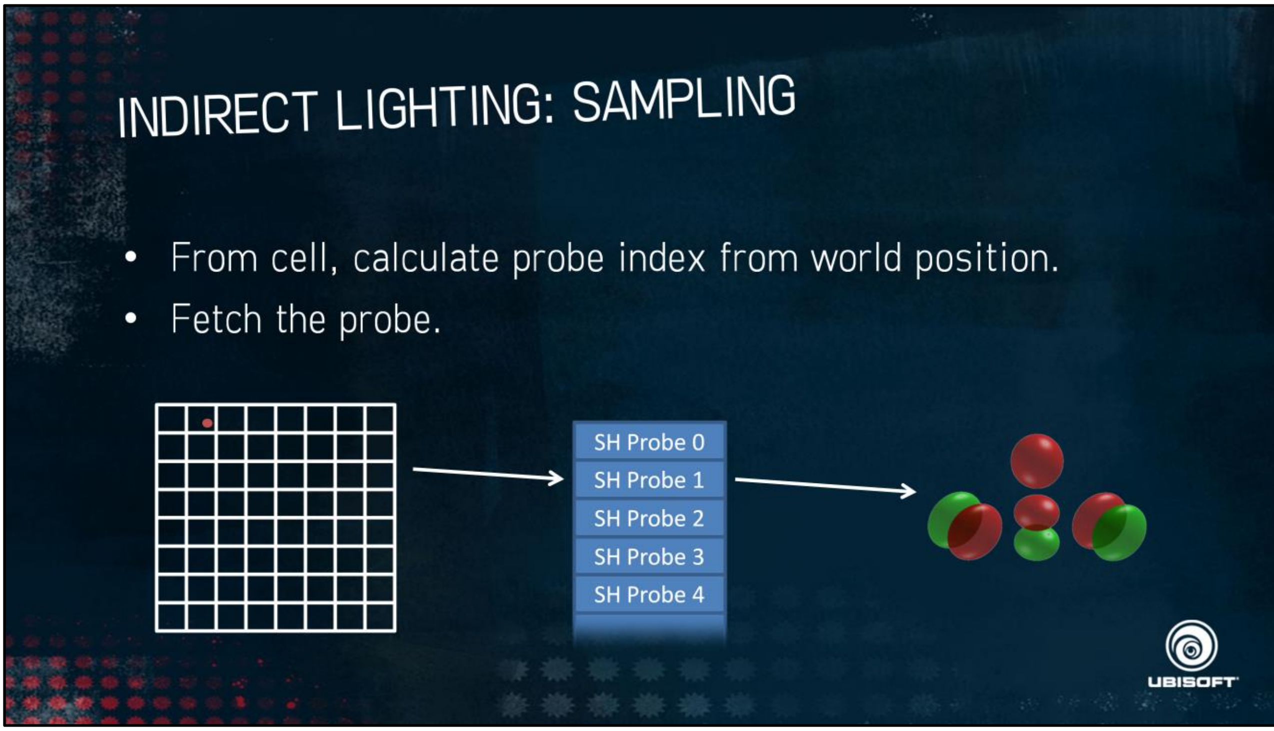
- Final page table is 160x160 sectors (highest mip resolution):

INDIRECT LIGHTING: SAMPLING

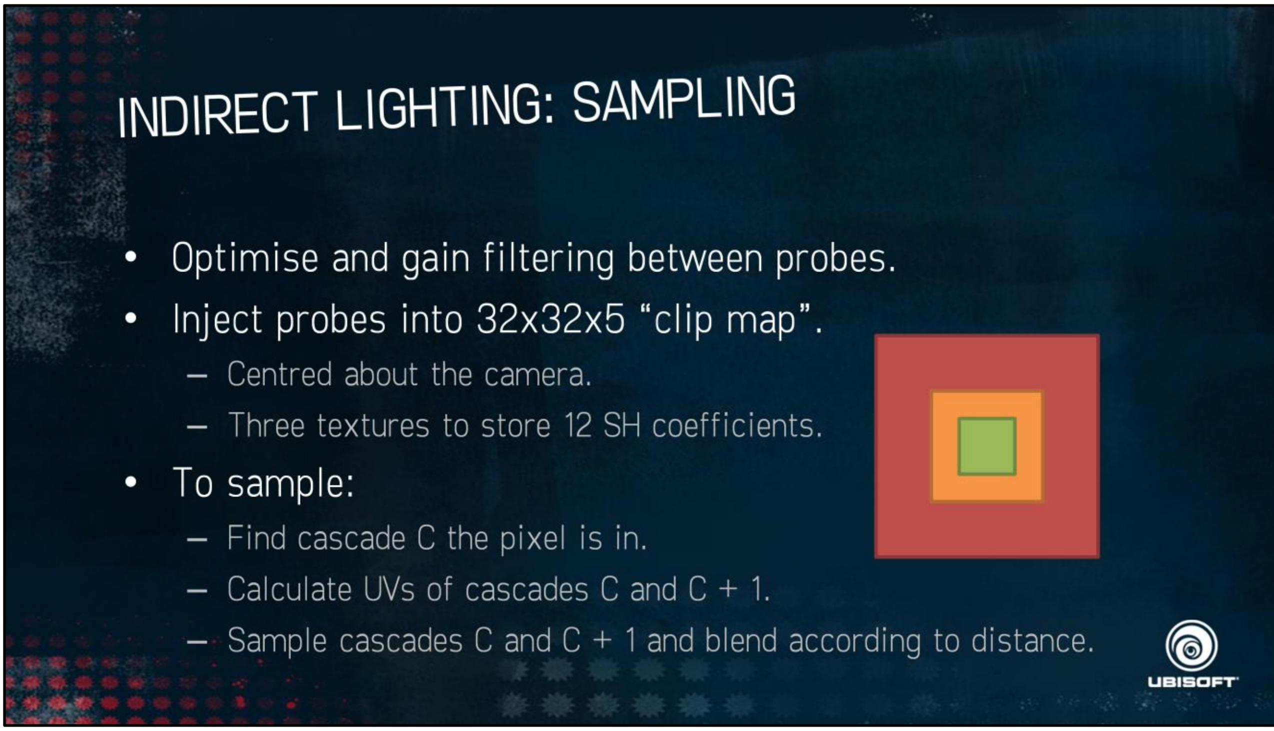
- On GPU, sample page table to find index into cell list:



UBISOFT

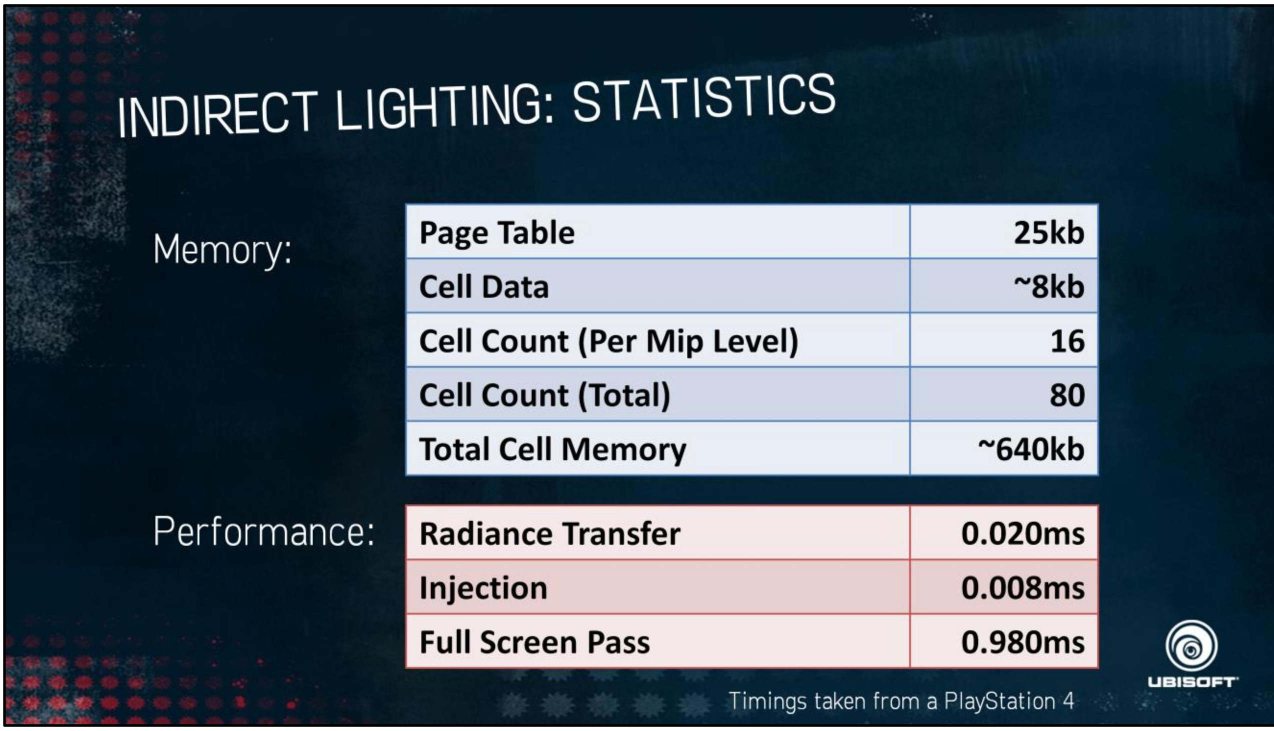


But how would we interpolate between probes? We'd have to do four expensive taps, which is why instead we inject into a clip map…

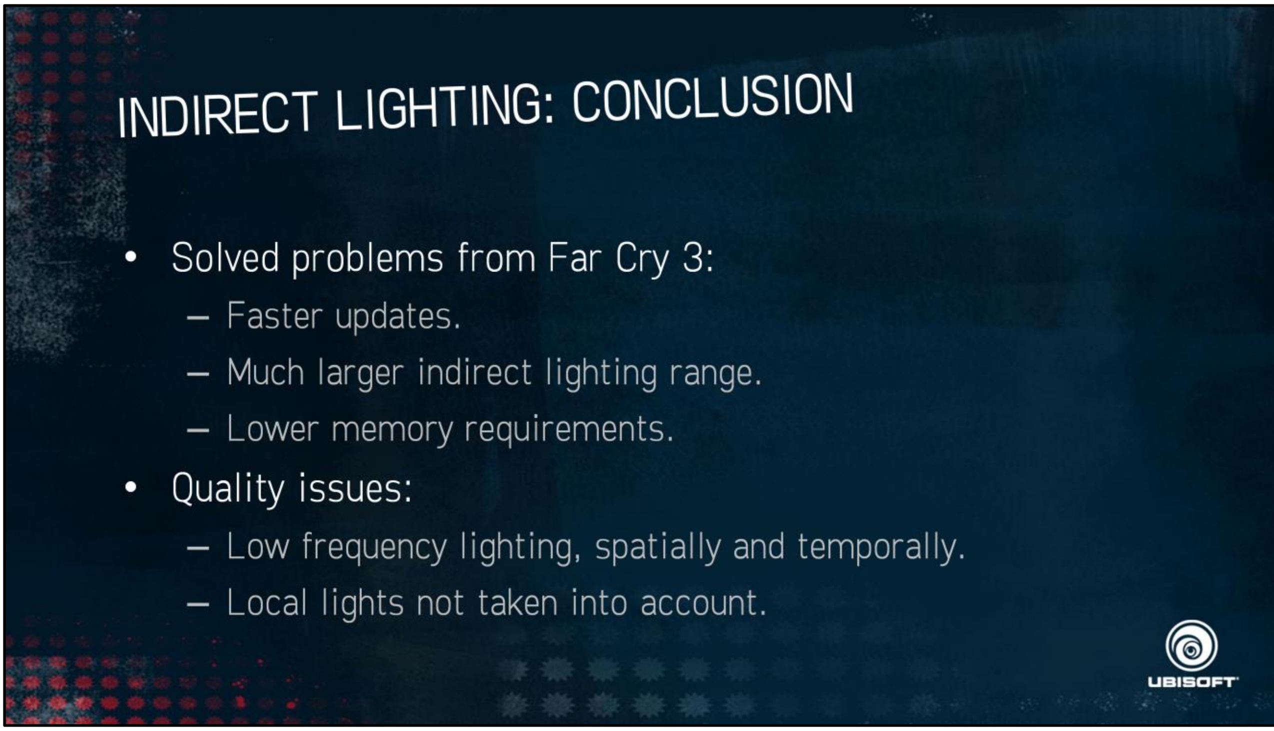


The clip map is a 32x32x5 texture array – each “mip level” needs to be the same size as the one above as it covers a larger area.

For last-gen, we just used a single level of this clip map to replace our old volume map which required a LOT of memory.



The sky lighting and indirect lighting full screen pass obviously takes the most time.
It’s currently VGPR bound.



The quality issues are a pretty big deal for us – a light probe every 8m is clearly not enough, and 2nd order radiance transfer just can't capture high frequency GI data. However, with our restrictions of using the same data across console generations, we did a good job of solving the problems we faced on Far Cry 3.

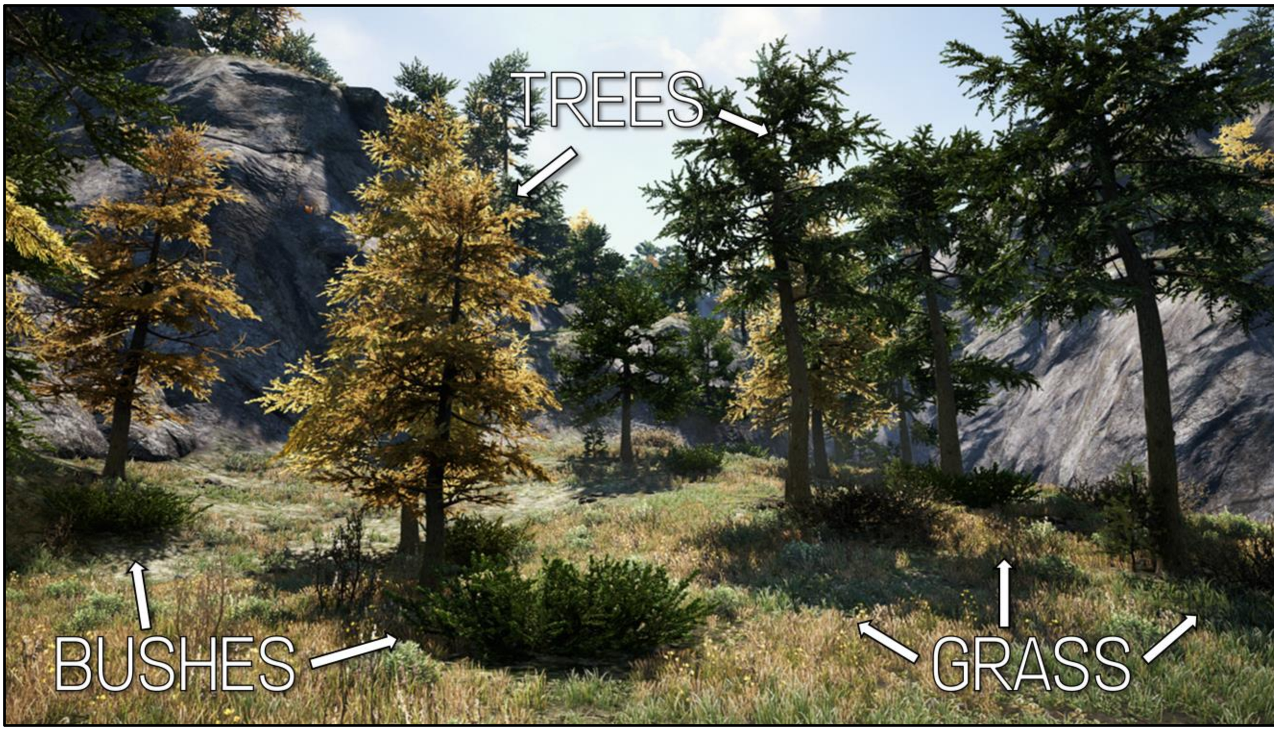
You might be wondering why we went for lower memory requirements than Far Cry 3, given the increased memory of current-gen hardware… well, we managed to use the memory optimisations on last-gen too, which saved us a few megabytes.


Rendering the World of Far Cry 4
VEGETATION

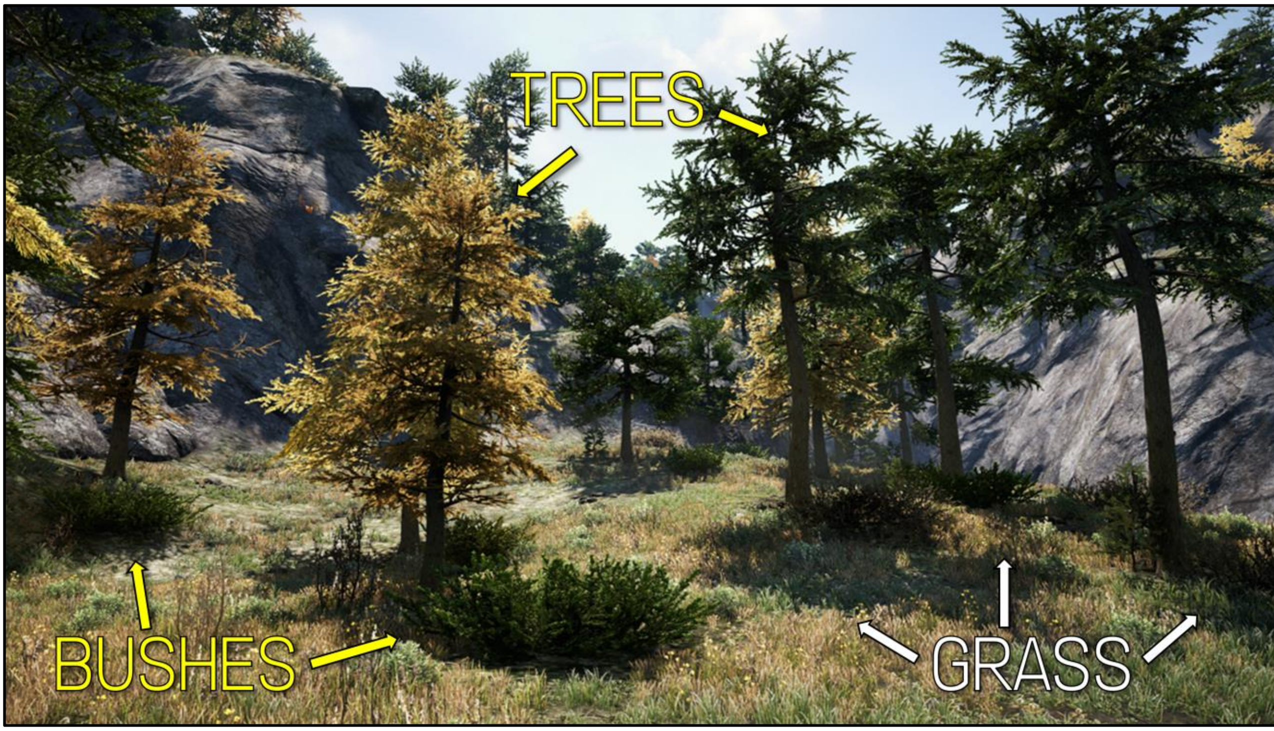
VEGETATION

- Major rendering focus on Far Cry 4.
- Completely different data set for current-gen.
- Goals:
  - Visual fidelity up close.
  - Improved LODding and impostors.
  - Simulation.

Credit to Philippe Gagnon and Jean-Sebastien Guay for developing this system.



Grass also covers small plants



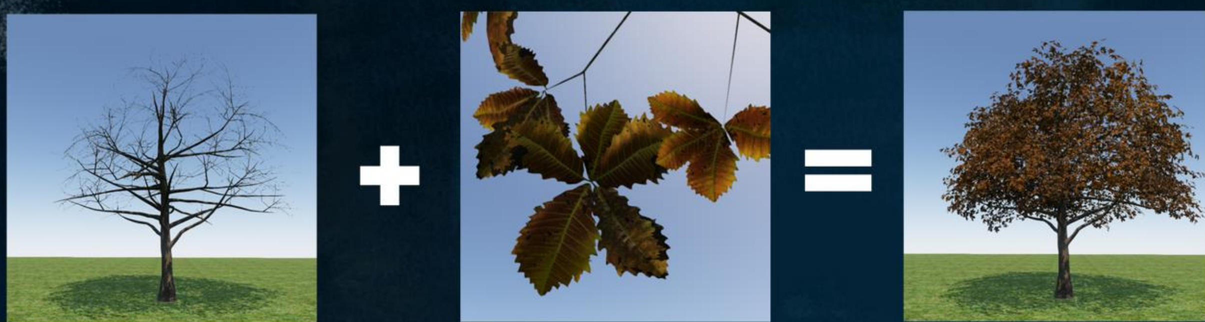
Grass also covers small plants

VEGETATION

- New vegetation system focused on trees and bushes.
- Create trees using SpeedTree and import into engine:

VEGETATION

- Trees consist of a trunk plus leaf clusters:



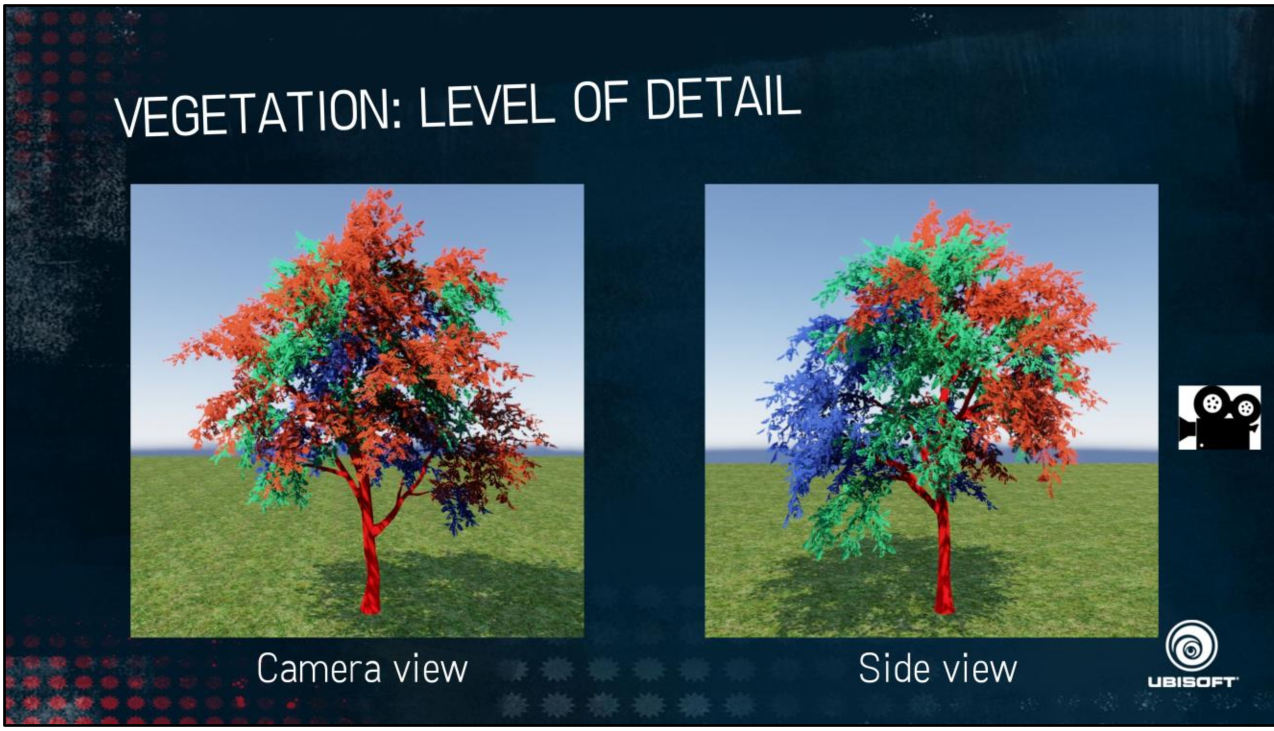
Everything is rendered with alpha test; no alpha blending is used.

VEGETATION: LEVEL OF DETAIL

- Trunks and leaf clusters have three LODs before a tree becomes an impostor.
- Leaf cluster LODding generated offline:
  - Calculate percentage of cluster visible:
    - 12 distances.
    - 9 viewpoints.

LOD 0
LOD 1
LOD 2

UBISOFT

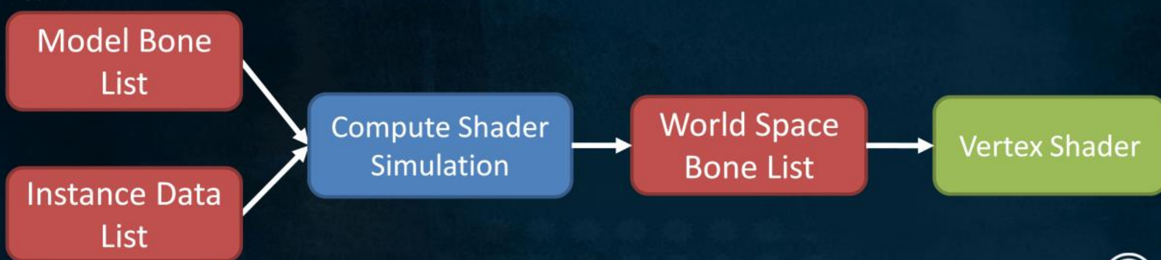


If we fix our culling camera then look around the side, we'll see that leaf clusters at the back of the tree have lower LODs.

Obviously, please remember that we generate these leaf cluster LODs not just at various viewpoints around the camera, but also at different distances too, so far away the whole tree would turn blue.

VEGETATION: SIMULATION

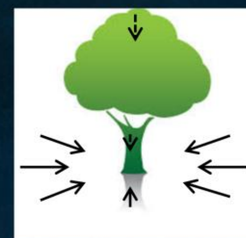
- Physics and movement simulated on GPU.
- Instanced bones transformed into world space bones.



Video of the buzzer. You can see how the wind from the buzzer affects the trees, bushes and also the grass too. Although the grass didn't have simulation, we used something very similar to our water ripple simulation. I'll be going over some similar hacks we did like that to put the finishing touches on the vegetation in a few minutes.

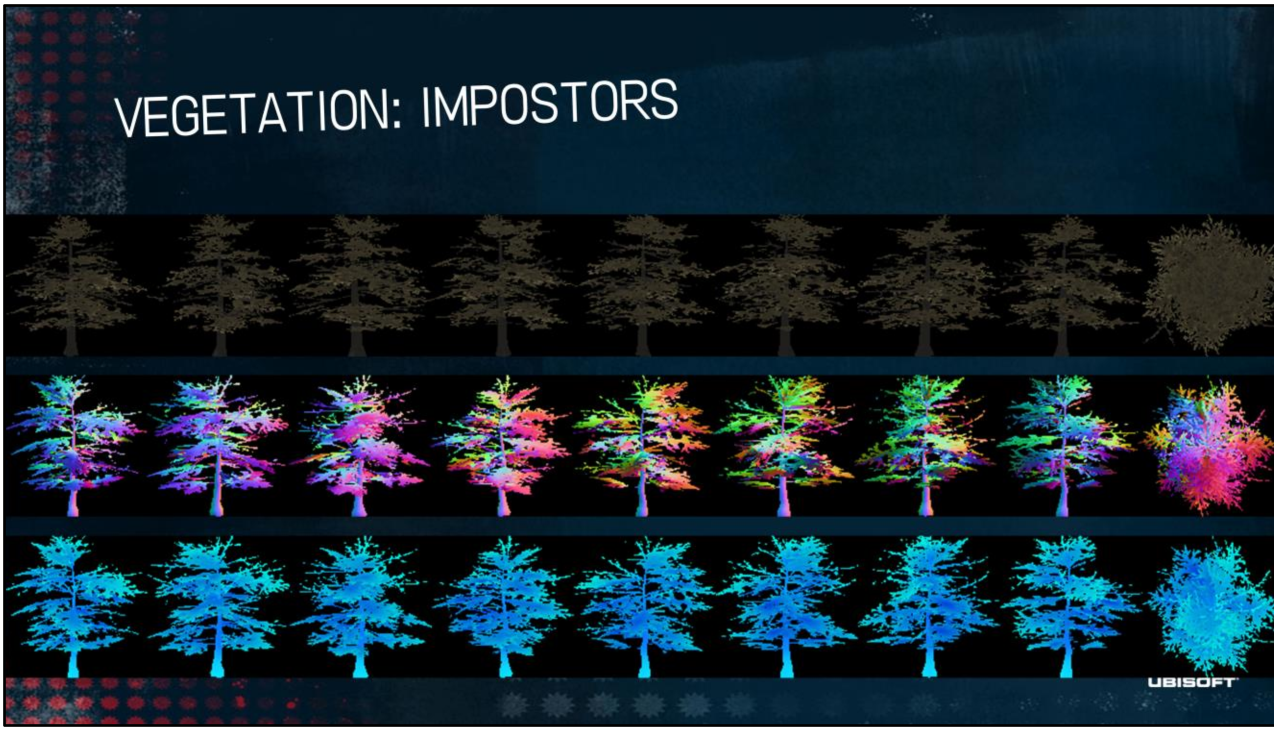
VEGETATION: IMPOSTORS

- Take screenshots from nine viewpoints:
  - Eight perpendicular to the tree.
  - One top down.



VEGETATION: IMPOSTORS

- “G-Buffer” screenshots:
- Capture albedo, normals and material properties:

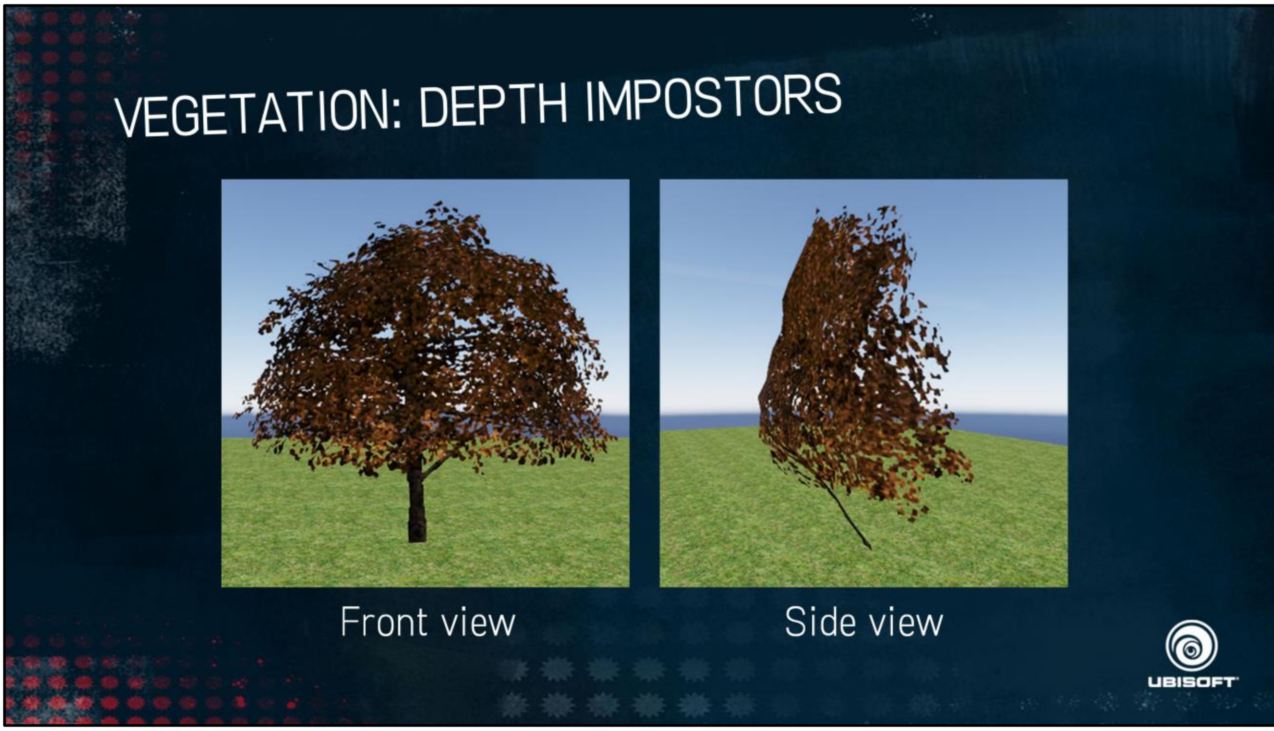


The problem with this impostor system is the high memory requirement caused by the number of textures. We might look at reducing the number of views in the future.

VEGETATION: IMPOSTORS

- Depth billboards:
  - Billboard geometry is tessellated into a 16x16 grid.
  - Capture depth during “G-Buffer” screenshots.
  - Displace vertices according to original tree depth.

UBISOFT



Although not demonstrated here, this really helps when a tree is lit from the side, or if two distant tree impostors intersect.

VEGETATION: AMBIENT OCCLUSION

- Generate AO volumes :
  - Range from 16x16x16 to 64x64x64 in size.
  - Capture shadow maps from 32 directions around the volume.
  - Sample shadows and average.

UBISOFT

VEGETATION: STATISTICS

	Larch	Rosewood
Trunk (LOD 0)	5,564	3,115
Leaves (LOD 0)	44,574	169,076
Leaves (LOD 1)	5,465	45,025
Leaves (LOD 2)	846	10,170

UBISOFT

It’s only possible to render this many vertices because we have the LODding system that we do. We aggressively cull high resolution LODs. Realistically, during rendering, the rosewood would have around 80,000 vertices max.

VEGETATION: PERFORMANCE

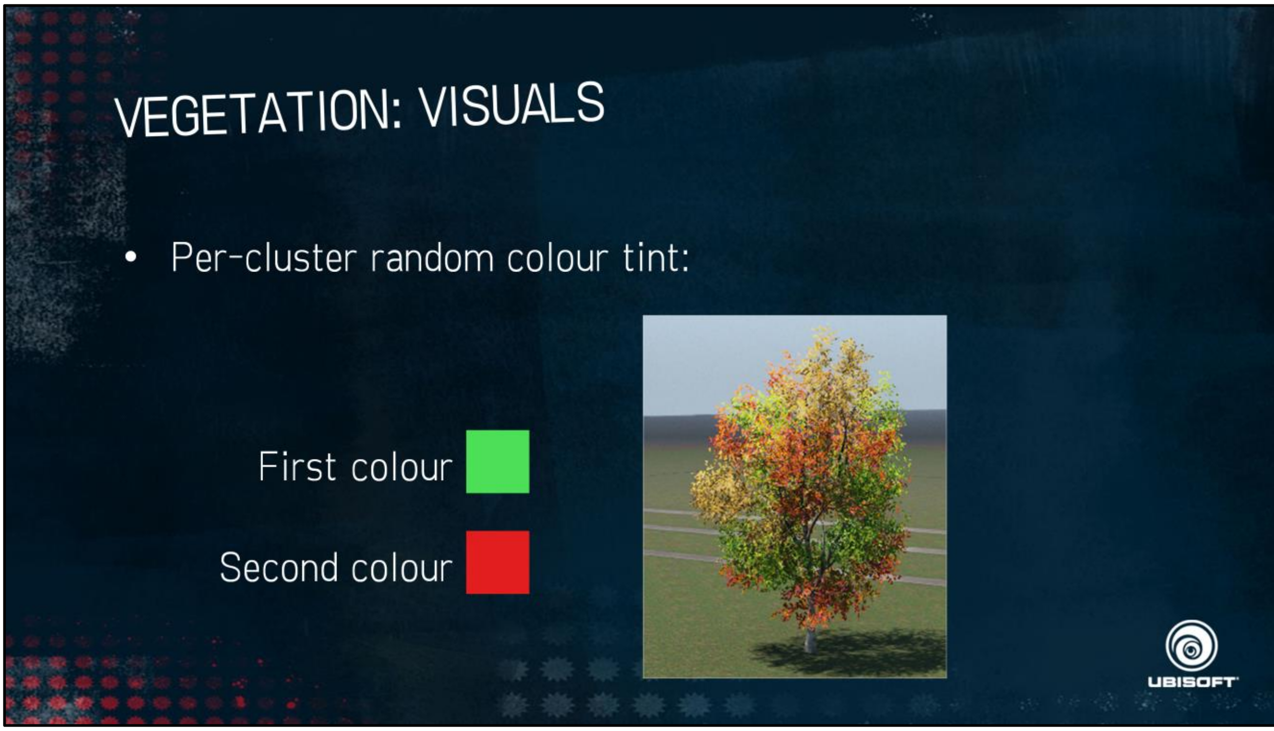
Tree Simulation	**0.28ms**
Shadow Pass	**0.95ms**
G-Buffer Pass	**5.50ms**

Timings taken from a PlayStation 4

VEGETATION: VISUALS

- Visually, vegetation is tricky to get right.
- Not simulating many things:
  - Light bounces between blades of grass.
  - Light scattering through leaves.
  - etc.
- Wizardry from technical artists required.

UBISOFT



This is simple, but very effective. Obviously, this is an extreme example but it’s not too different to trees I’ve seen in Vermont in the autumn I guess.

VEGETATION: VISUALS

- Bringing impostors to life:
  - Per-vertex sine waves to simulate branch motion.
    - Easy with tessellated billboards.
    - Simulated by smooth triangle waves for speed. [Sousa2007]
    - Waves in X, Y and Z for macro detail.
    - Waves in X and Z for micro detail.

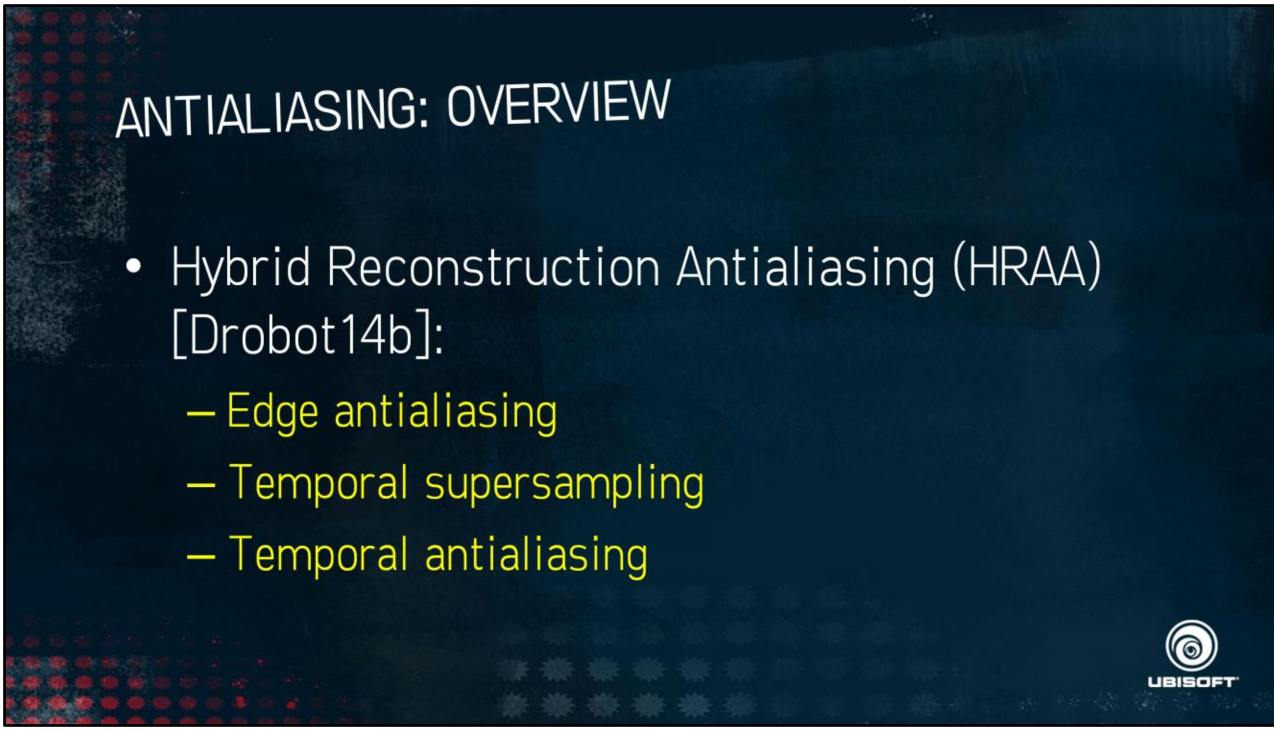
VEGETATION: VISUALS

- Bringing impostors to life:
  - Per-pixel noise from texture to simulate leaf motion.

No noise on the left, noise on the right.

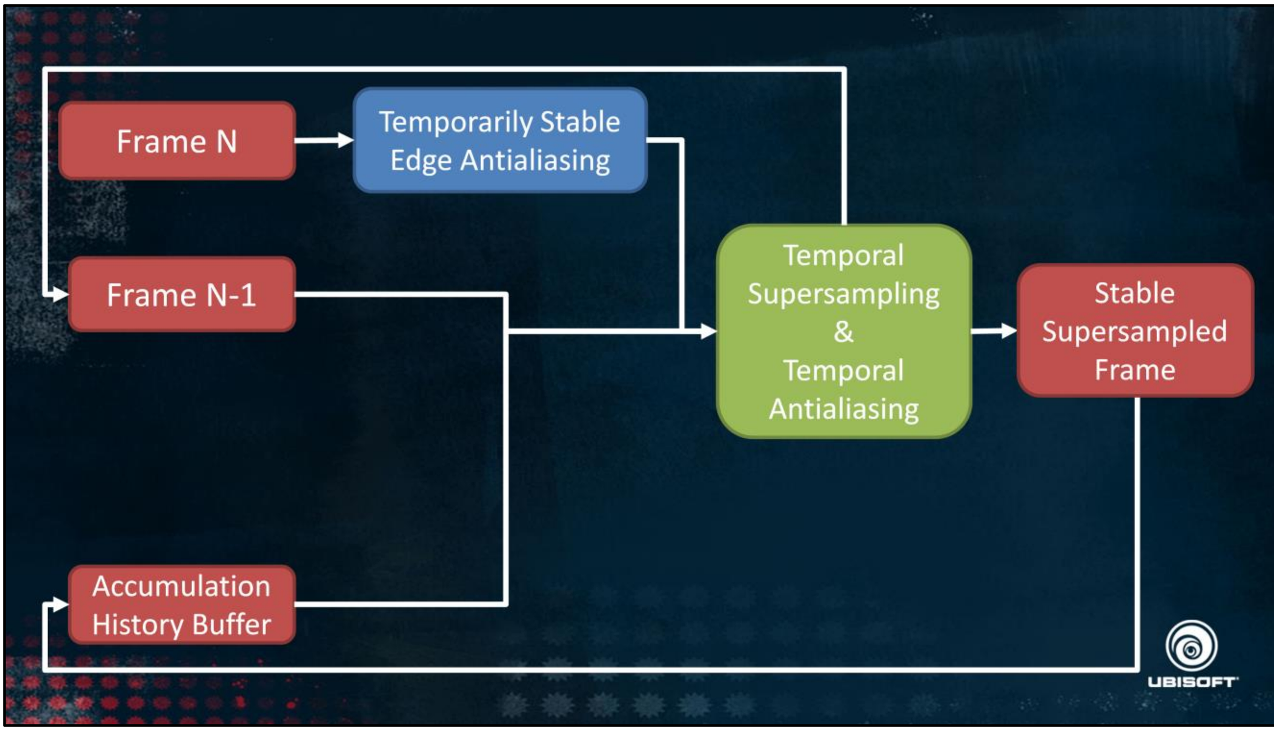
Video of noise on/off.


Rendering the World of Far Cry 4
ANTIALIASING



Our antialiasing approach was presented by Michal Drobot at SIGGRAPH 2014, so please read that presentation for many more details. And of course, full credit (and also all the difficult questions) should go to Michal for the work he put into this. It consists of a combination of three techniques…

Edge antialiasing – this should be self-explanatory.
Temporal antialiasing – this refers to aliasing between two successive frames – we'd like to make this look smooth too.
Temporal supersampling – supersampling is rendering at an increased resolution – we'd like to do that temporally, by sampling different pixels of the larger image each frame.

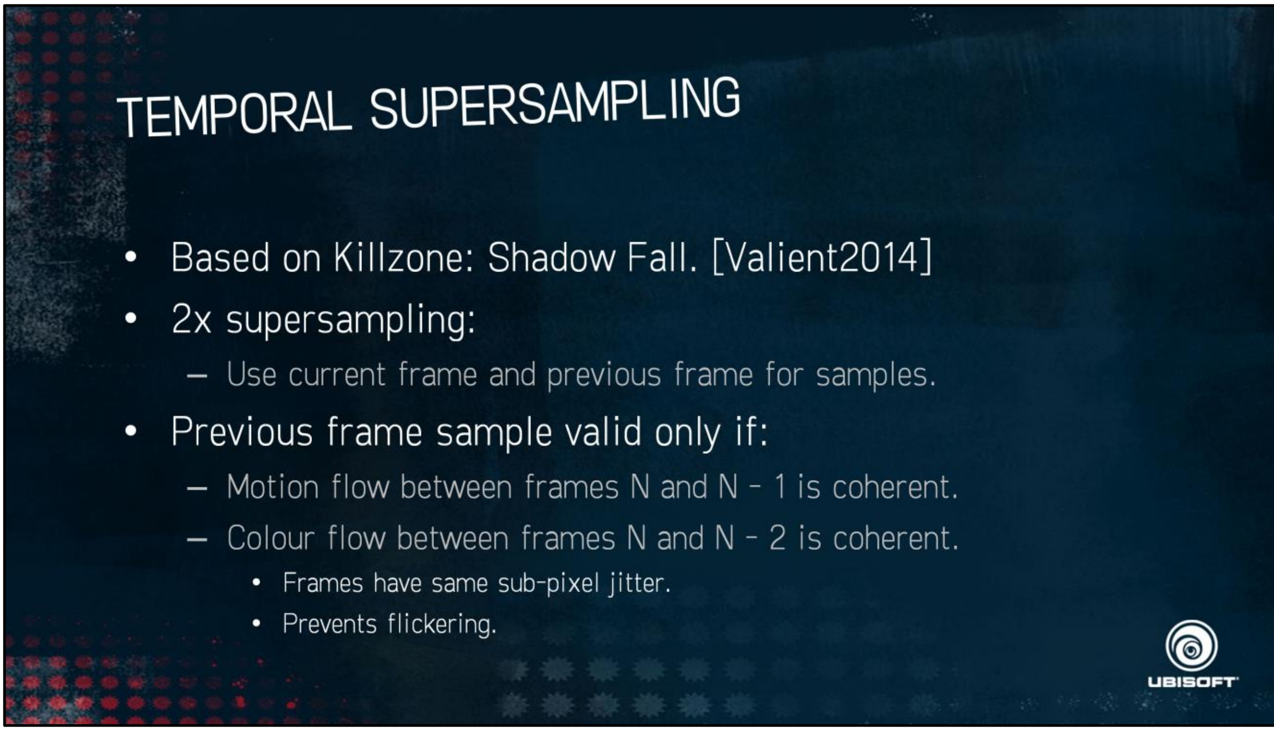


Here’s an overview of what’s going on… don’t worry… we’ll break it down over the next few minutes.

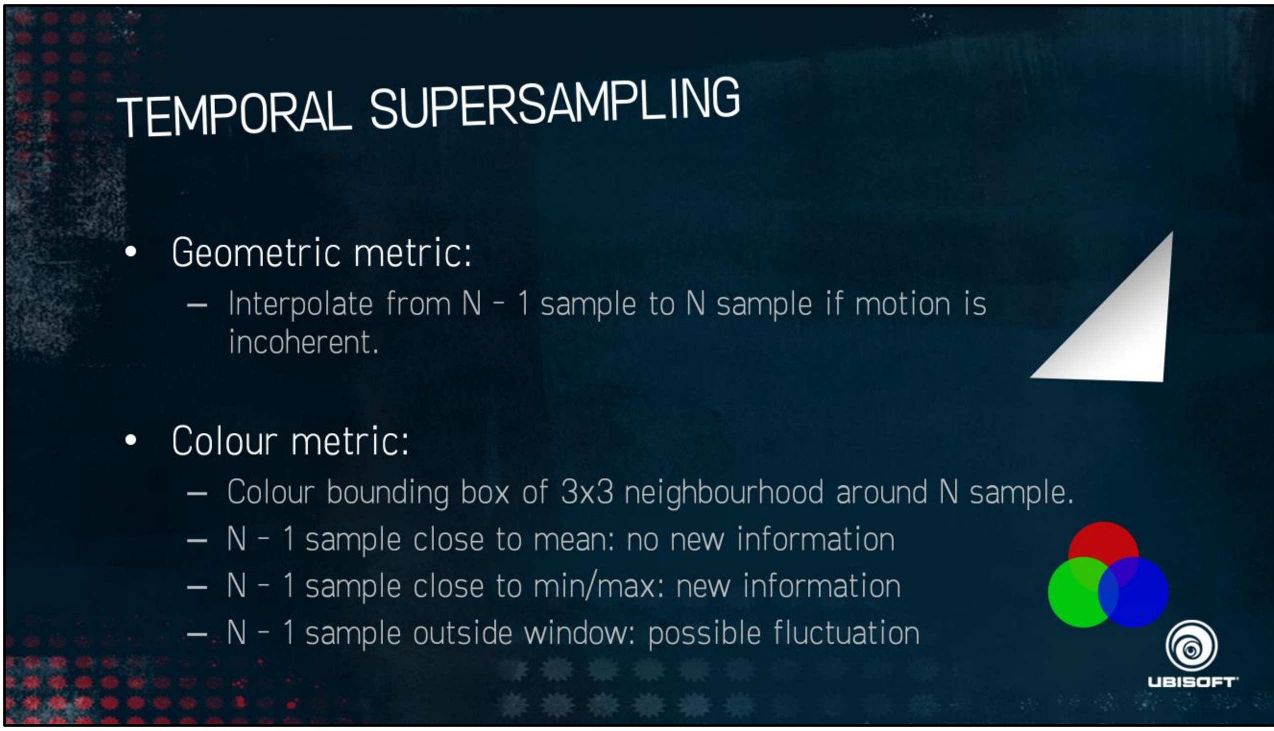
EDGE ANTIALIASING

- SMAA:
  - Temporally stabilised.
  - Normal, depth and luma predicated thresholding.
- AEAA (Analytical Edge Antialiasing):
  - For alpha tested geometry. [Persson2011]
- CRAA (Coverage Reconstruction Antialiasing):
  - See [Drobot2014b].
  - Best performance but content issues.

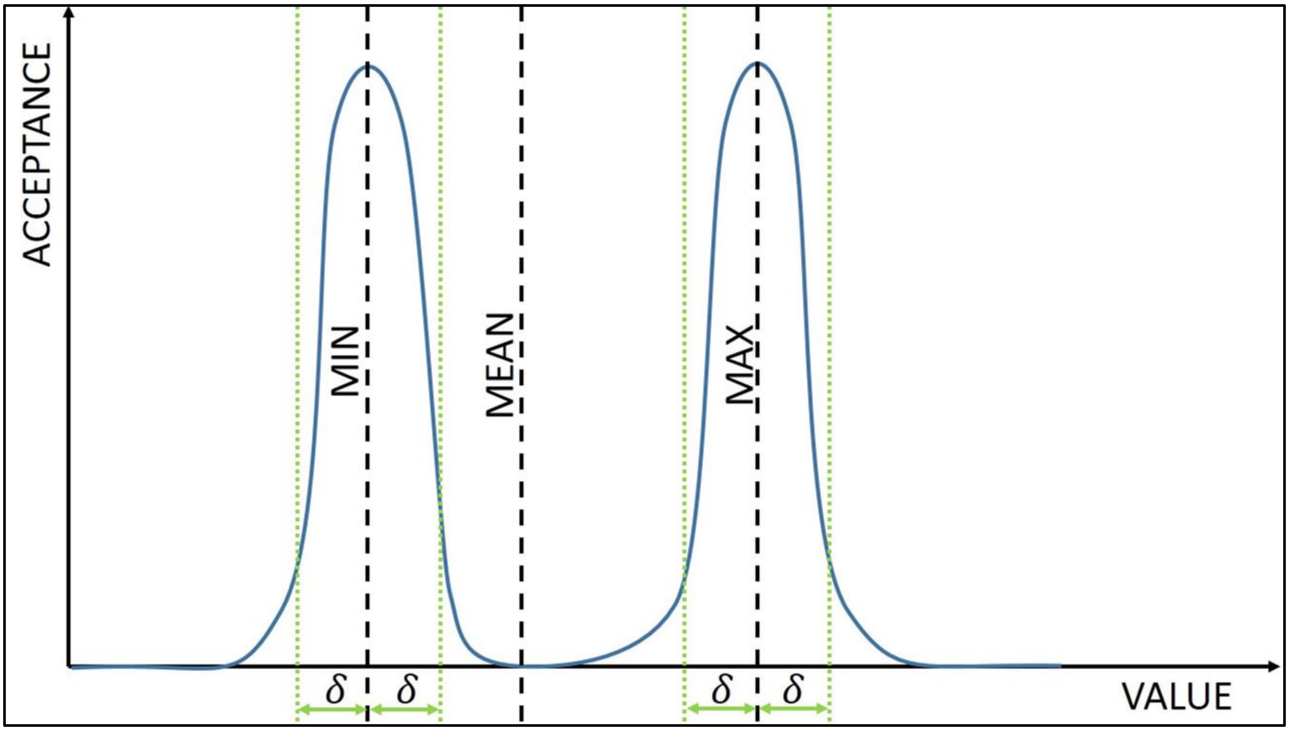
UBISOFT



We didn’t ship Far Cry 4 with colour flow coherency, hence we have some flickering present in the game.



Actually used centre pixel of 3x3 window rather than the mean, because that resulted in too many false positives due to too much smoothing. It’s also really worth reading Brian Karis’ excellent talk from SIGGRAPH 2014 where he discusses other various acceptance metrics.

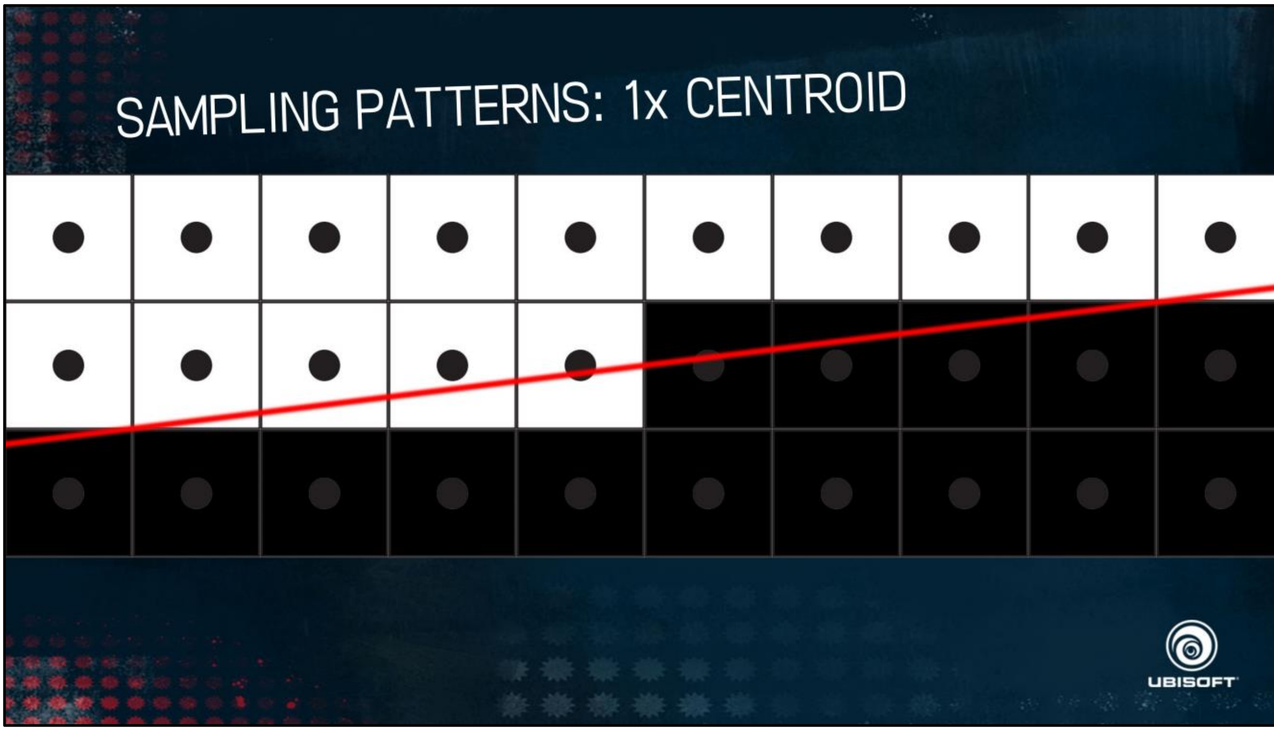


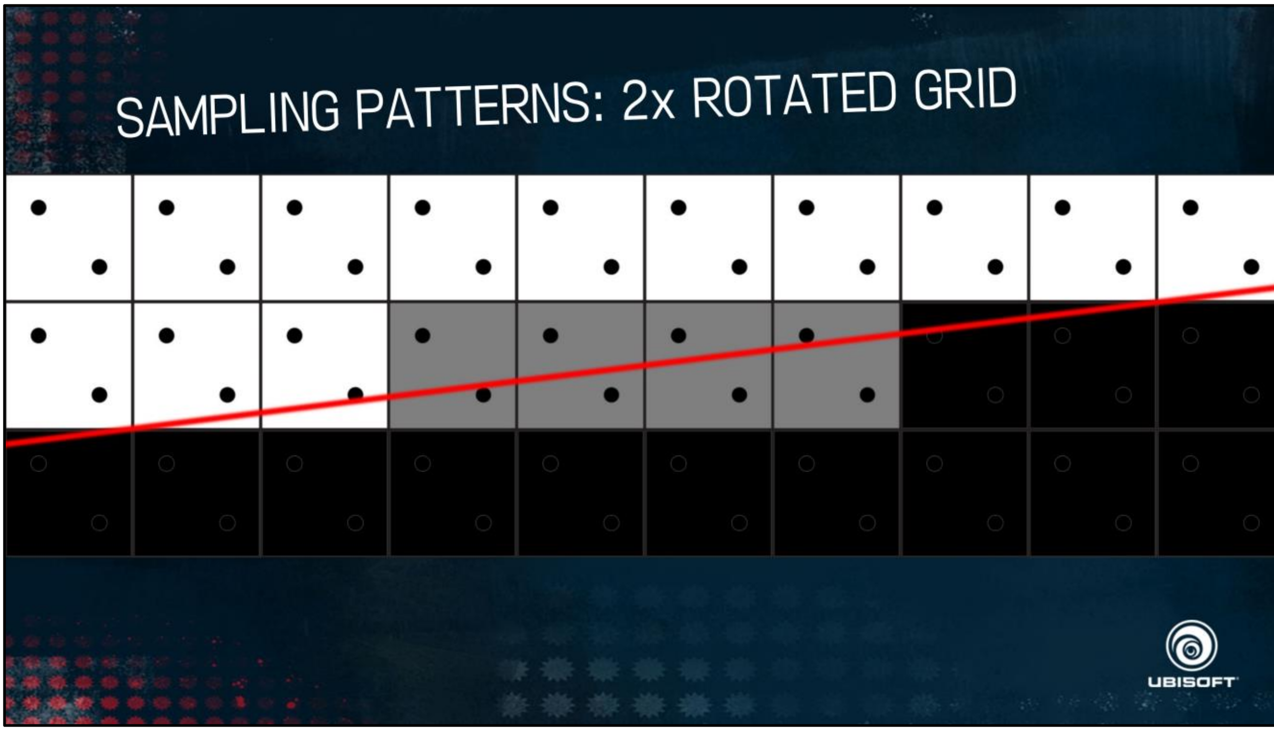
So this is a visualisation of our acceptance metric, which I hope makes things a lot clearer. Again, we use the centre pixel rather than the mean.

TEMPORAL SUPERSAMPLING

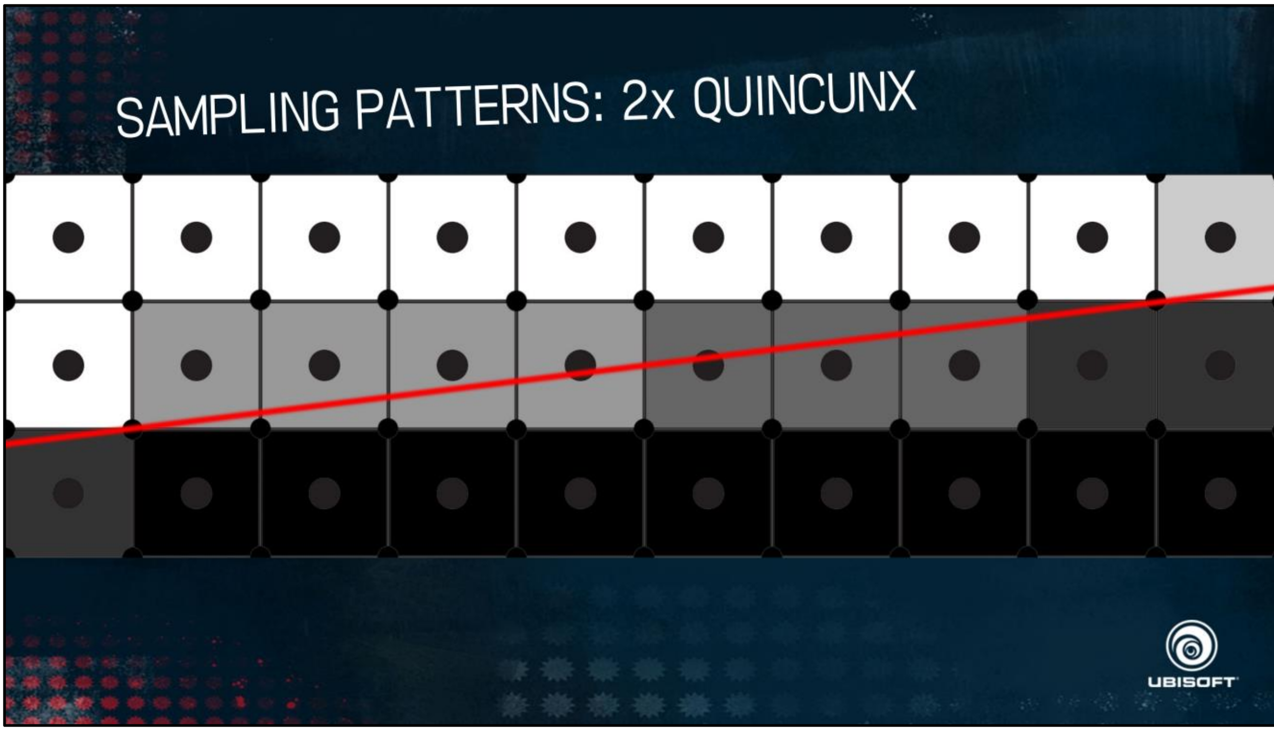
- Maximise our two samples for best image.
- Various sampling patterns:
  - 1x Centroid
  - 2x Rotated Grid
  - 2x Quincunx
  - 4x Rotated Grid
  - 2x FLIPQUAD

UBISOFT

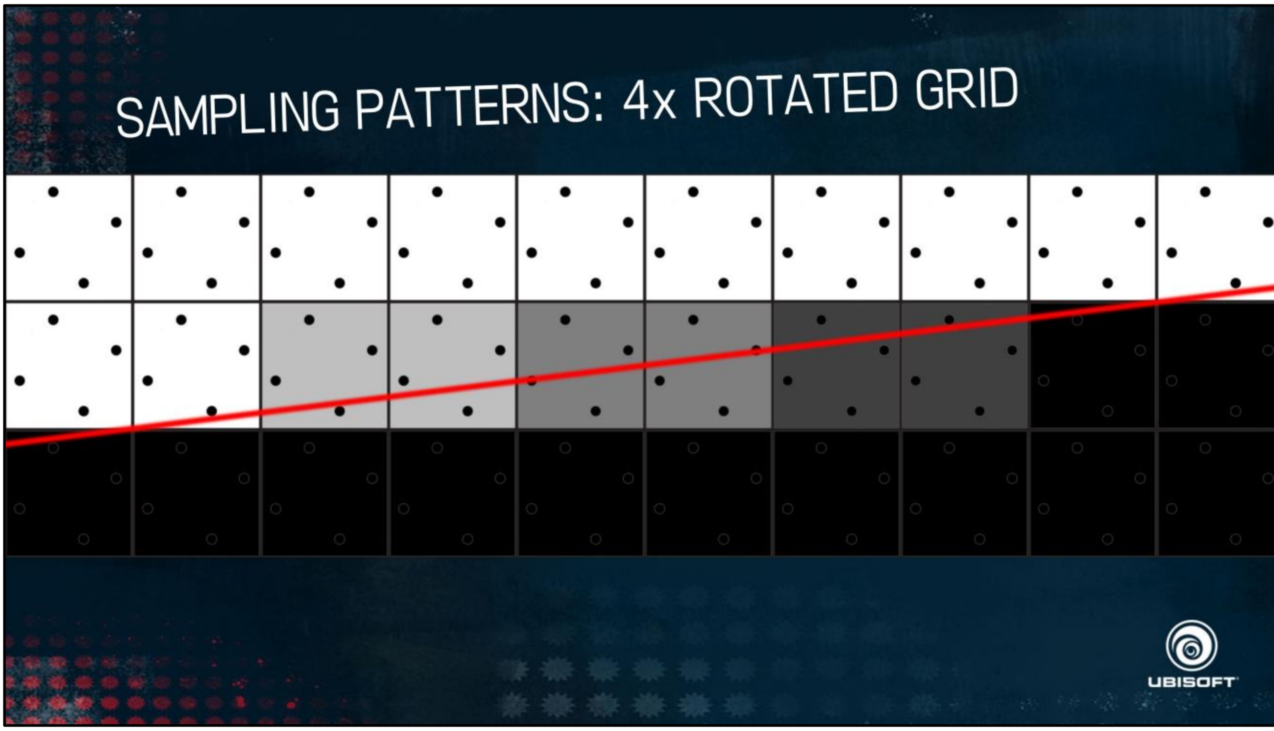

SAMPLING PATTERNS: 1x CENTROID
UBISOFT



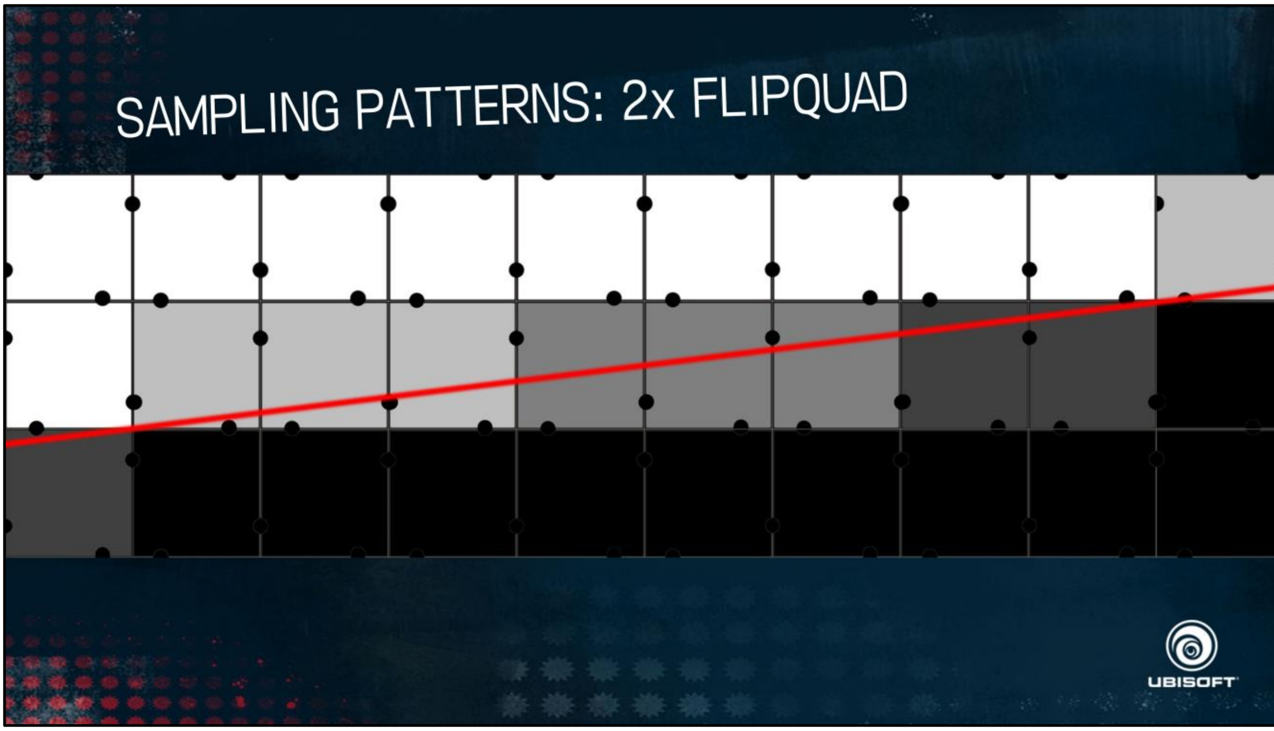
2xRG has 2 unique columns and 2 unique rows.



QUINCUNX optimizes the pattern by sharing corner samples with adjacent pixels.
It covers 3 unique rows and 3 unique columns, improving over 2xRG.
It adds a 0.5 radius blur (that is partially recoverable by 0.5 pixel unsharp mask processing).



4xRG has 4 unique columns and 4 unique rows.



FLIPQUAD is a efficient 2 sample / pixel scheme that allows effective 4x Super-sampling by sharing sampling points on pixel boundary edges.
It combines the benefits of QUINCUNX and Rotated Grid patterns.
Covers 4 unique rows and 4 unique columns, improving over 2xRG and QUINCUNX, matching 4xRG
It adds a 0.5 radius blur (that is partially recoverable by 0.5 pixel unsharp mask processing).

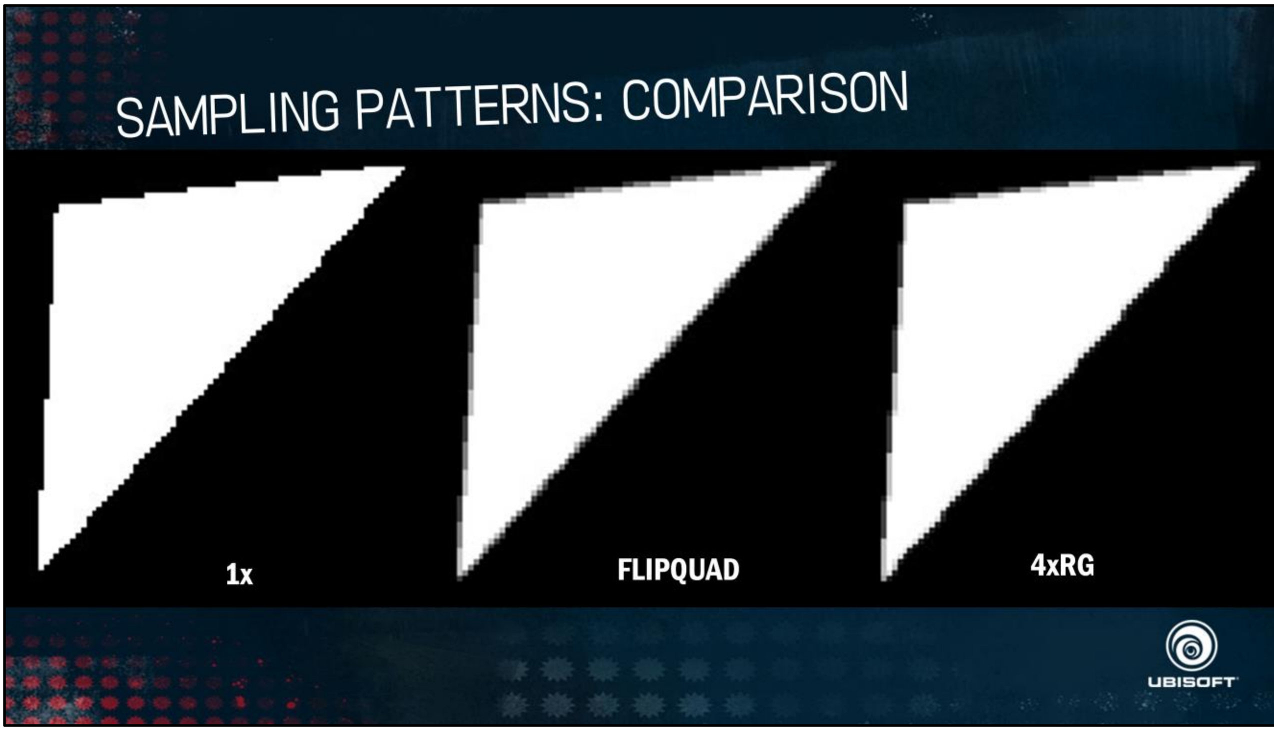


Image courtesy [Akenine 03]
We can see FLIPQUAD performing similar to 4xRotated Grid.
It can provide arguably higher quality results.

[T]EMPORAL SUPERSAMPLING

- FLIPQUAD gives significantly higher quality than QUINCUNX at same cost [Laine2006]:

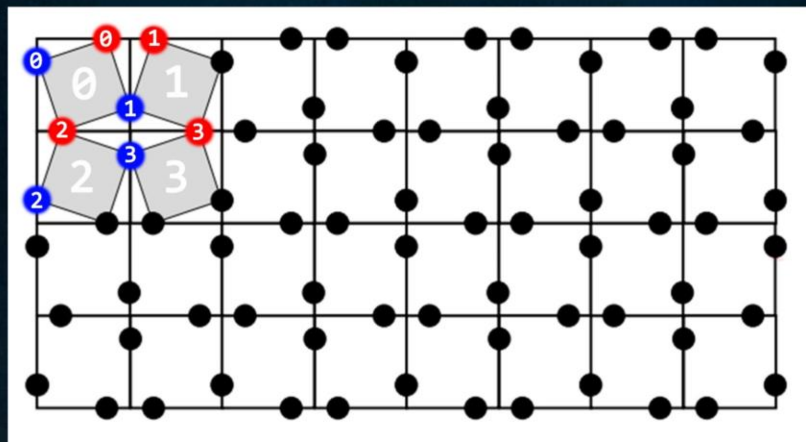
Pattern	E
1x Centroid	>1.0
4x Uniform Grid	0.698
4x Rotated Grid	0.439
Quincunx	0.518
FLIPQUAD	0.364

UBISOFT

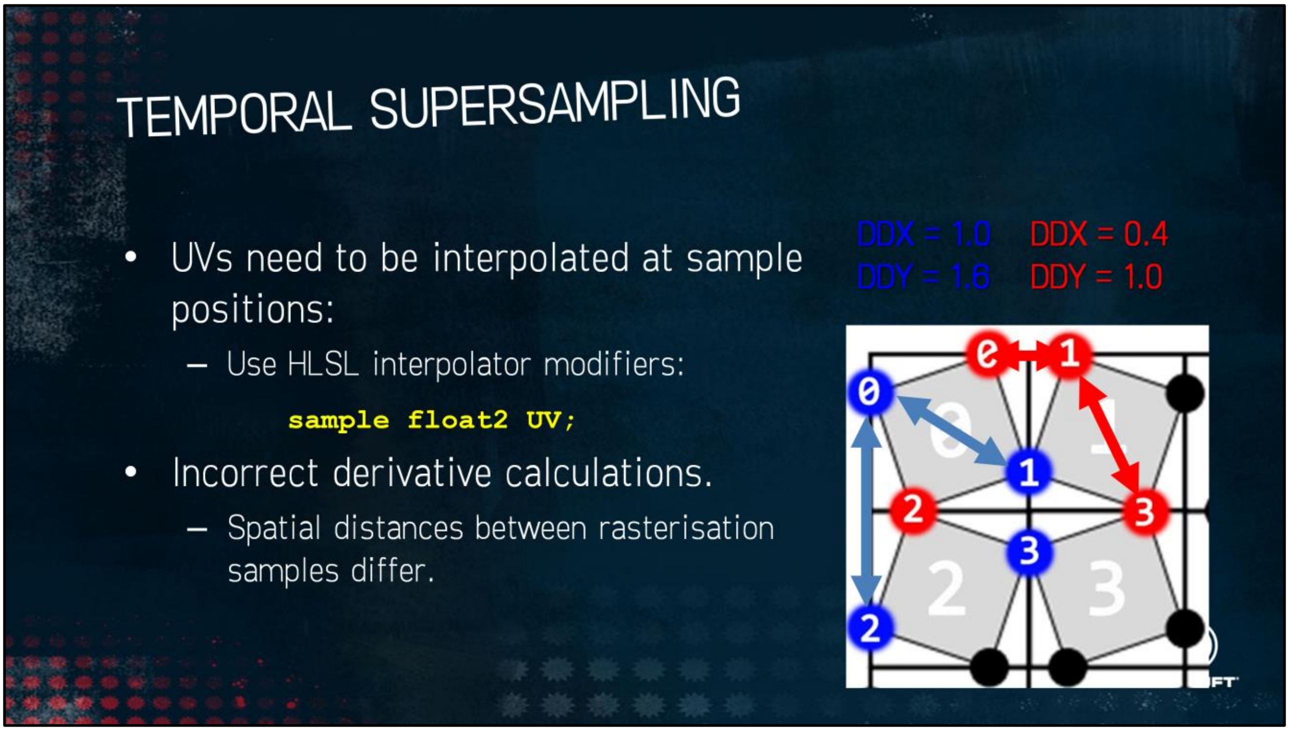
Lower error estimate E -> closer to 1024 super-sampled reference.
Image courtesy [Laine 06]

TEMPORAL SUPERSAMPLING

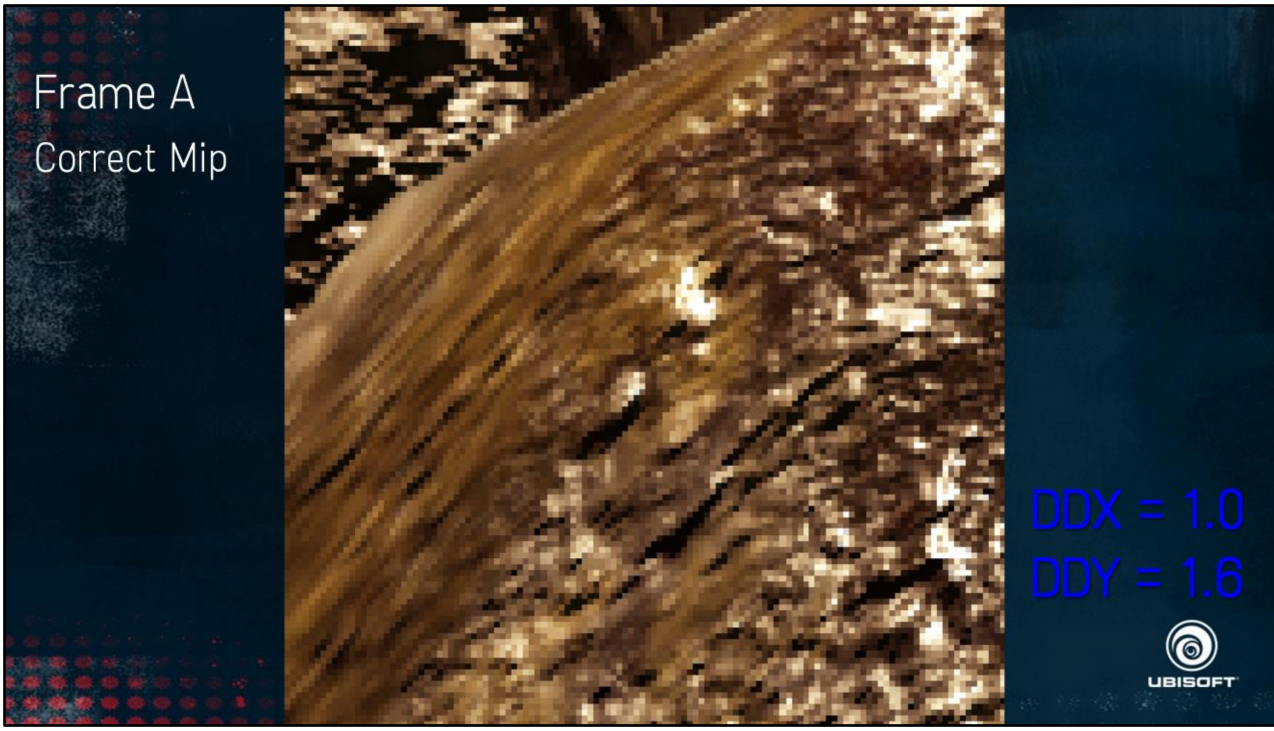
- Split the pattern in half:
  - Frame A
  - Frame B
- Pixel #0 = Average(0, 1, 0, 2)



UBISOFT

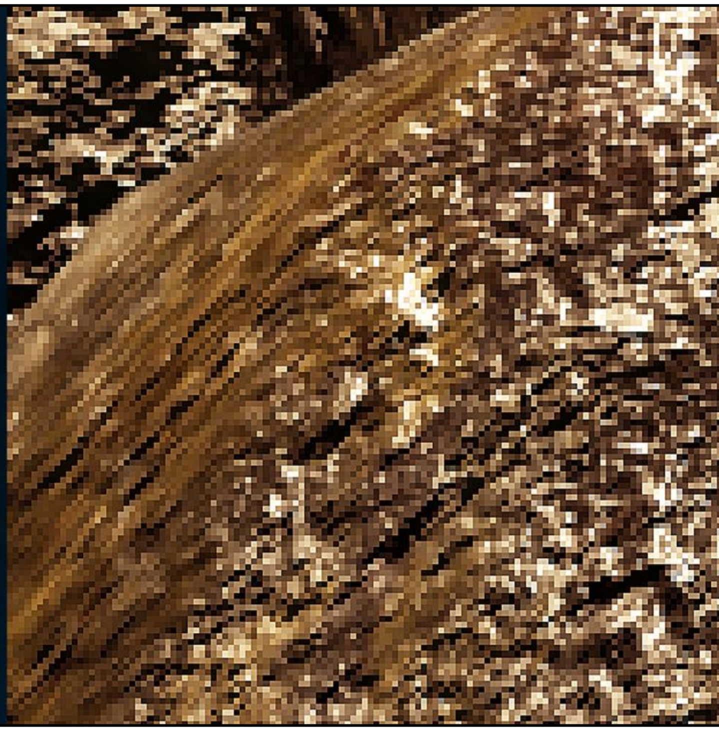


Of course, this all sounds like a good idea but there are problems in practice… you have to interpolate UVs at sample positions to achieve actual supersampling, then you find that the derivative calculations are incorrect. This causes nasty problems like the following…


Frame A
Correct Mip
DDX = 1.0
DDY = 1.6
UBISOFT

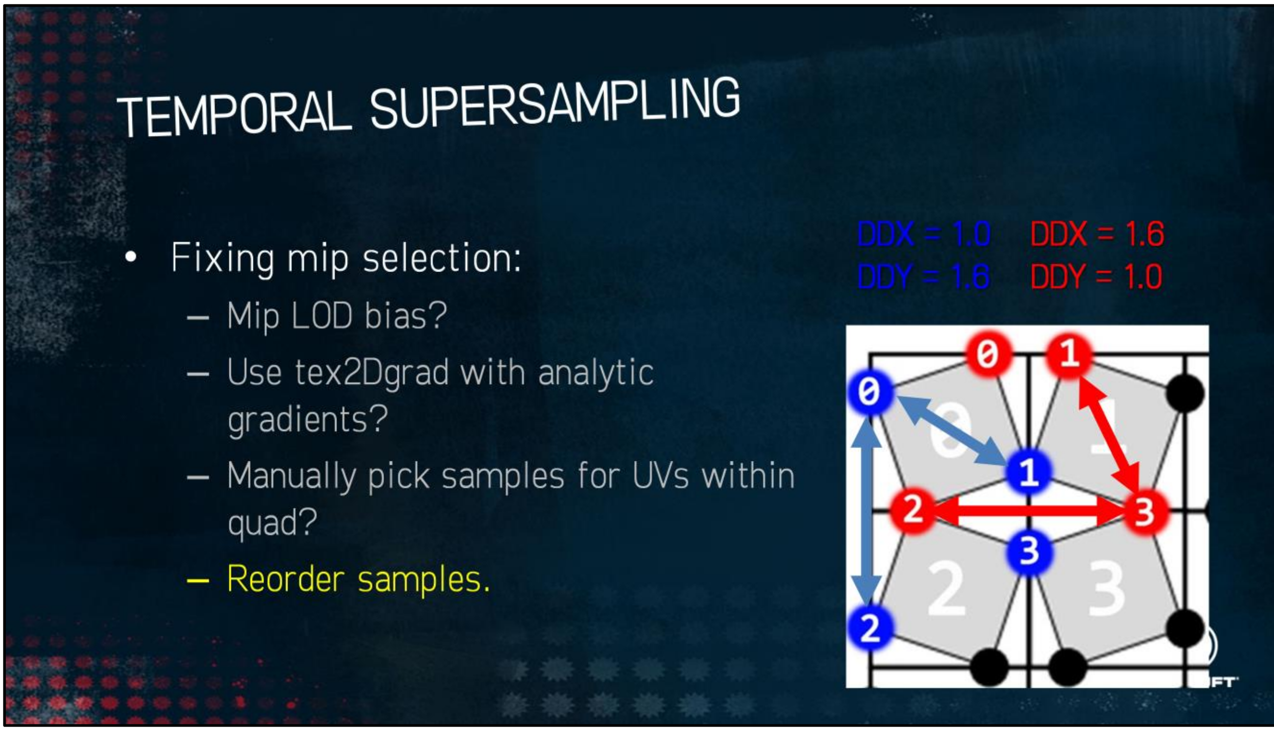
Frame B

Oversharpened
Mip



DDX = 0.4
DDY = 1.0

UBISOFT

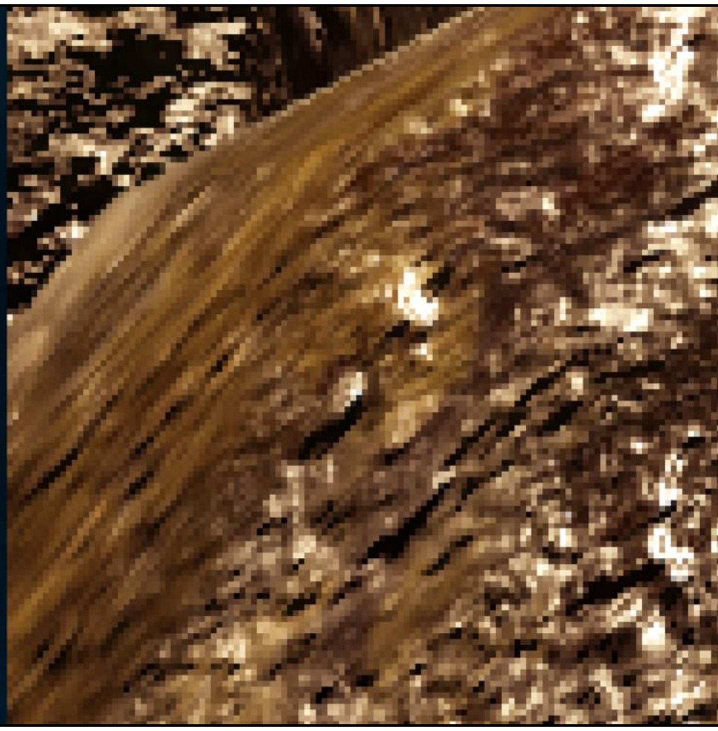


If we just reorder the samples so that samples 2 and 3 in the red frame are in fact samples 0 and 1, then the gradients are similar.

Frame A

Correct Mip

Reordered Samples



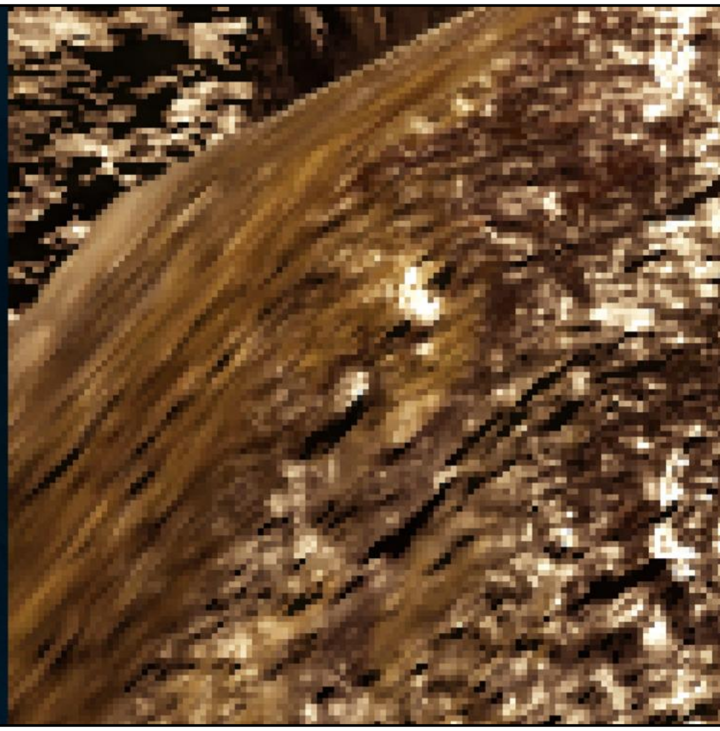
DDX = 1.0

DDY = 1.6

Frame B

Correct Mip

Reordered Samples



DDX = 1.6

DDY = 1.0

UBISOFT

TEMPORAL ANTIALIASING

- Temporally stabilise many of our buffers:
  - Final frame buffer
  - SMAA
  - SSBC
  - Motion vectors
  - Bloom

UBISOFT

TEMPORAL ANTIALIASING

- History exponential buffer:
  - Amortize sudden visual changes.
  - Accumulate new “important” data.
- Same acceptance metrics as temporal supersampling.
  - Geometric metric.
  - Colour metric.
  - Now applied to full history buffer as well as N – 1 frame.
  - Might need to tweak values separately.

HIGH FREQUENCY RECOVERY

- Naïve supersampling loses detail:
  - Wider, more complex kernel needed to preserve detail. [Burley2007]
  - FLIPQUAD gives 0.5 pixel blur.
- Recover high frequency detail.

HIGH FREQUENCY RECOVERY

- Approximate 4-tap sinc kernel by:
  - 0.5 pixel radius box blur (FLIPQUAD resolve)
  - 0.5 pixel radius unsharp masking
- Mip bias of -0.5 on all textures.
  - To match supersampled resolution.

ANTIALIASING: PERFORMANCE

Single Pass	Timing (ms)
Temporal FLIPQUAD (TFQ)	0.20
SMAA	1.00
TAA	0.60
TFQ + TAA	0.65
SMAA + TFQ + TAA	1.65
Unsharp Mask	0.28

Timings taken from a PlayStation 4


Illuminating Far Cry 4
CONCLUSION

PERFORMANCE

- Performance of a typical Far Cry 4 scene on GPU:

Shadow Maps	1.8ms
G-Buffer	7.5ms
Lighting	6.6ms
Blended	1.7ms
Post-Processing	2.8ms
Antialiasing	1.9ms
Miscellaneous	1.1ms

Timings taken from a PlayStation 4

UBISOFT

Yes, this scene is well in budget! It helps to be cross-generation sometimes.

CONCLUSION

- Significant rendering improvements whilst maintaining a last-gen build:
  - Anisotropic and metallic materials.
  - Higher fidelity sky occlusion and indirect lighting.
  - New vegetation system.
  - Hybrid Reconstruction Antialiasing.

UBISOFT

THANKS

- Far Cry 4 team:
  - Jeremy Moore
  - Philippe Gagnon
  - Michal Drobot
  - Jean-Sébastien Guay
  - Gabriel Lassonde
  - Sébastien Viard
  - Everyone else
- Special thanks:
  - Stephen Hill
  - Julien Merceron

REFERENCES

- [Burley2012] Physically Based Shading at Disney, Brent Burley, SIGGRAPH 2012, http://blog.selfshadow.com/publications/s2012-shading-course/
- [Chen2015] Adaptive Virtual Texture Rendering in Far Cry 4, GDC 2015
- [Colbert2008] GPU-Based Importance Sampling, Mark Colbert and Jaroslav Křivánek, GPU Gems 3, http://http.developer.nvidia.com/GPUGems3/gpugems3_ch20.html
- [Drobot2014a] Low Level Optimizations for GCN, Digital Dragons 2014, http://michaldrobot.files.wordpress.com/2014/05/gcn_alu_opt_digitaldragons2014.pptx
- [Drobot2014b] Hybrid Reconstruction Anti-Aliasing, SIGGRAPH 2014, http://advances.realtimerendering.com/s2014/drobot/hraa.pptx
- [Frey2011] Spherical Skinning with Dual-Quaternions and Qtangents, Ivo Zoltan Frey, SIGGRAPH 2011, http://www.crytek.com/cryengine/presentations/spherical-skinning-with-dual-quaternions-and-qtangents
- [Frykholm2009] The BitSquid Low-Level Animation System, Niklas Frykholm, http://bitsquid.blogspot.ca/2009/11/bitsquid-low-level-animation-system.html
- [Hill2012] Rock-Solid Shading, Stephen Hill & Dan Baker, SIGGRAPH 2012, http://blog.selfshadow.com/publications/
- [Lagarde2014] Moving Frostbite to PBR, Sébastien Lagarde and Charles de Rousiers, SIGGRAPH 2014

UBISOFT

REFERENCES

- [Laine2006] A Weighted Error Metric and Optimization Method for Antialiasing Patterns, Laine and Aila, Computer Graphics Forum 25(1), http://research.nvidia.com/sites/default/files/publications/laine2006cgf_paper.pdf
- [Lazarov2011] Physically Based Lighting in Call of Duty: Black Ops, Dimitar Lazarov, SIGGRAPH 2011, http://advances.realtimerendering.com/s2011/Lazarov-Physically-Based-Lighting-in-Black-Ops%20(Siggraph%202011%20Advances%20in%20Real-Time%20Rendering%20Course).pptx
- [McAuley2012] Calibrating Lighting and Materials in Far Cry 3, Stephen McAuley, SIGGRAPH 2012, http://blog.selfshadow.com/publications/s2012-shading-course/mcauley/s2012_pbs_farcry3_notes_v2.pdf
- [Persson2011] Geometry Buffer Antialiasing, Emil Persson, SIGGRAPH 2011, http://iryoku.com/aacourse/downloads/11-Geometry-Buffer-Anti-Aliasing-%28GBAA%29.pdf
- [Revie2011] Implementing Fur in Deferred Shading, Donald Revie, GPU Pro 2
- [Sousa2007] Vegetation Procedural Animation and Shading in Crysis, Tiago Sousa, GPU Gems 3, http://http.developer.nvidia.com/GPUGems3/gpugems3_ch16.html
- [Stefanov2012] Deferred Radiance Transfer Volumes, Nikolay Stefanov & Mickael Gilabert, GDC 2012, http://www.gdcvault.com/play/1015326/Deferred-Radiance-Transfer-Volumes-Global
- [Valient2014] Taking Killzone: Shadow Fall Image Quality into the Next Generation, Michal Valient, GDC 2014, http://ams-cms/publications/presentations/GDC2014_Valient_Killzone_Graphics.pdf

UBISOFT


QUESTIONS?
UBISOFT


Rendering the World of Far Cry 4
APPENDIX A

QUATERNION: FROM TANGENT SPACE

```
float4 TangentSpaceToQuaternion(float3 tangent, float3 binormal, float3 normal)
{
    float4 quaternion;
    quaternion.x = normal.y - binormal.z;
    quaternion.y = tangent.z - normal.x;
    quaternion.z = binormal.x - tangent.y;
    quaternion.w = 1.0f + tangent.x + binormal.y + normal.z;

    return normalize(quaternion);
}
```

UBISOFT

QUATERNION: TO TANGENT SPACE

```
void QuaternionToTangentSpace(in float4 quaternion,
                              out float3 tangent,
                              out float3 binormal,
                              out float3 normal)
{
    tangent  = float3( 1.0f,  0.0f,  0.0f)
             + float3(-2.0f,  2.0f,  2.0f) * quaternion.y * quaternion.yxw
             + float3(-2.0f, -2.0f,  2.0f) * quaternion.z * quaternion.zwx;
    binormal = float3( 0.0f,  1.0f,  0.0f)
             + float3( 2.0f, -2.0f,  2.0f) * quaternion.z * quaternion.wzy
             + float3( 2.0f, -2.0f, -2.0f) * quaternion.x * quaternion.yxw;
    normal   = float3( 0.0f,  0.0f,  1.0f)
             + float3( 2.0f,  2.0f, -2.0f) * quaternion.x * quaternion.zwx
             + float3(-2.0f,  2.0f, -2.0f) * quaternion.y * quaternion.wzy;
}
```

UBISOFT

QUATERNION: UNPACKING FROM 10:10:10:2

```
float4 UnpackQuaternion(float4 packedQuaternion)
{
    uint index = (uint)(packedQuaternion.w * 3.0f);

    float4 quaternion;
    quaternion.xyz = packedQuaternion.xyz * sqrt(2.0f) - (1.0f / sqrt(2.0f));
    quaternion.w   = sqrt(1.0f - saturate(dot(quaternion.xyz, quaternion.xyz)));

    if(index == 0) quaternion = quaternion.wxyz;
    if(index == 1) quaternion = quaternion.xwyz;
    if(index == 2) quaternion = quaternion.xywz;

    return quaternion;
}
```

UBISOFT

QUATERNION: PACKING TO 10:10:10:2

```
float4 PackQuaternion(float4 quaternion)
{
    uint index = FindGreatestComponent(quaternion);

    if(index == 0) quaternion = quaternion.yzwx;
    if(index == 1) quaternion = quaternion.xzwy;
    if(index == 2) quaternion = quaternion.xywz;

    float4 packedQuaternion;
    packedQuaternion.xyz = quaternion.xyz * sign(quaternion.w) * sqrt(0.5f) + 0.5f;
    packedQuaternion.w   = index / 3.0f;

    return packedQuaternion;
}
```

UBISOFT