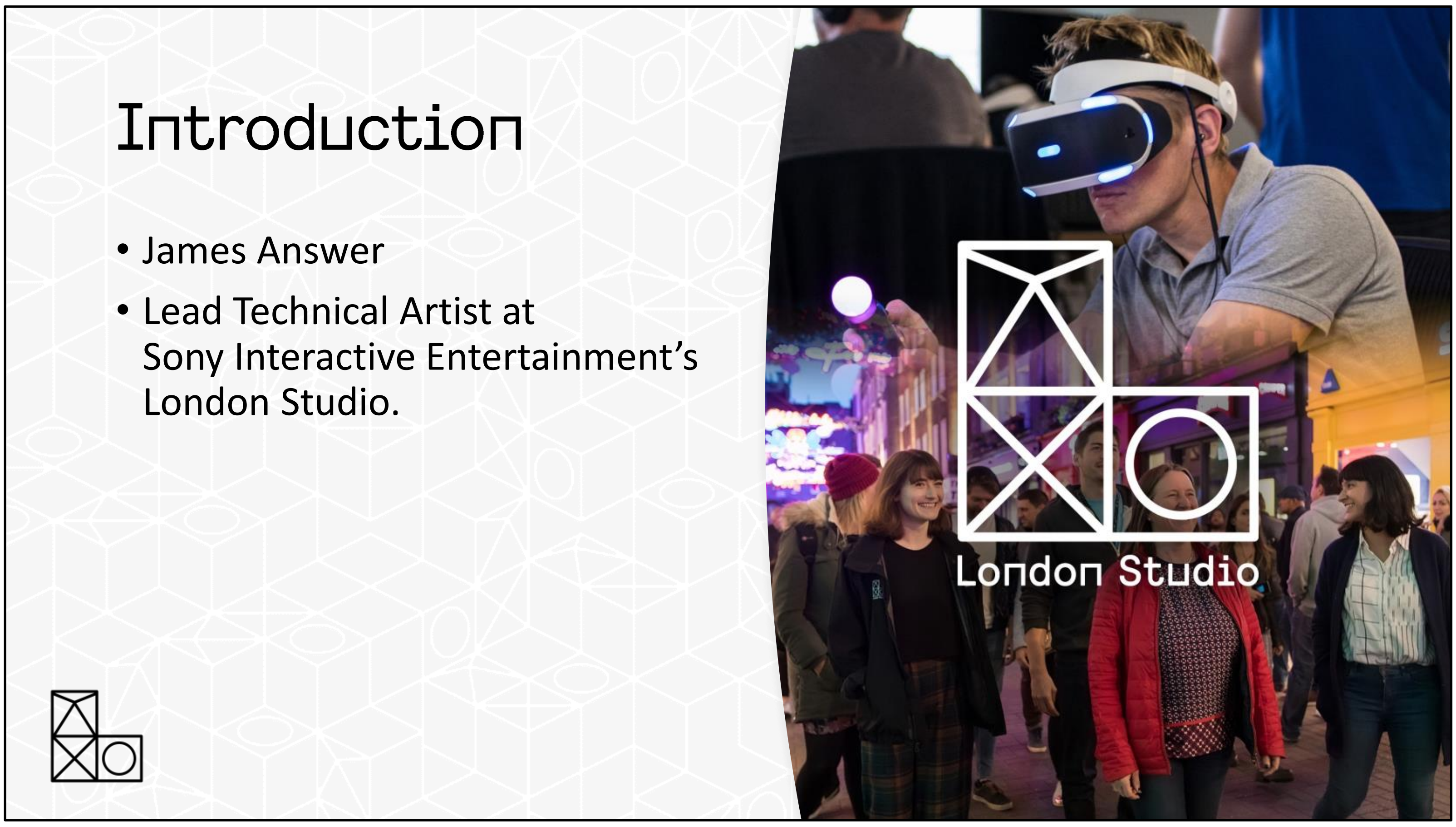
SIGGRAPH
THINK BEYOND
2020 S2020.SIGGRAPH.ORG

CHARACTER RENDERING FOR VR IN “BLOOD & TRUTH”

James Answer



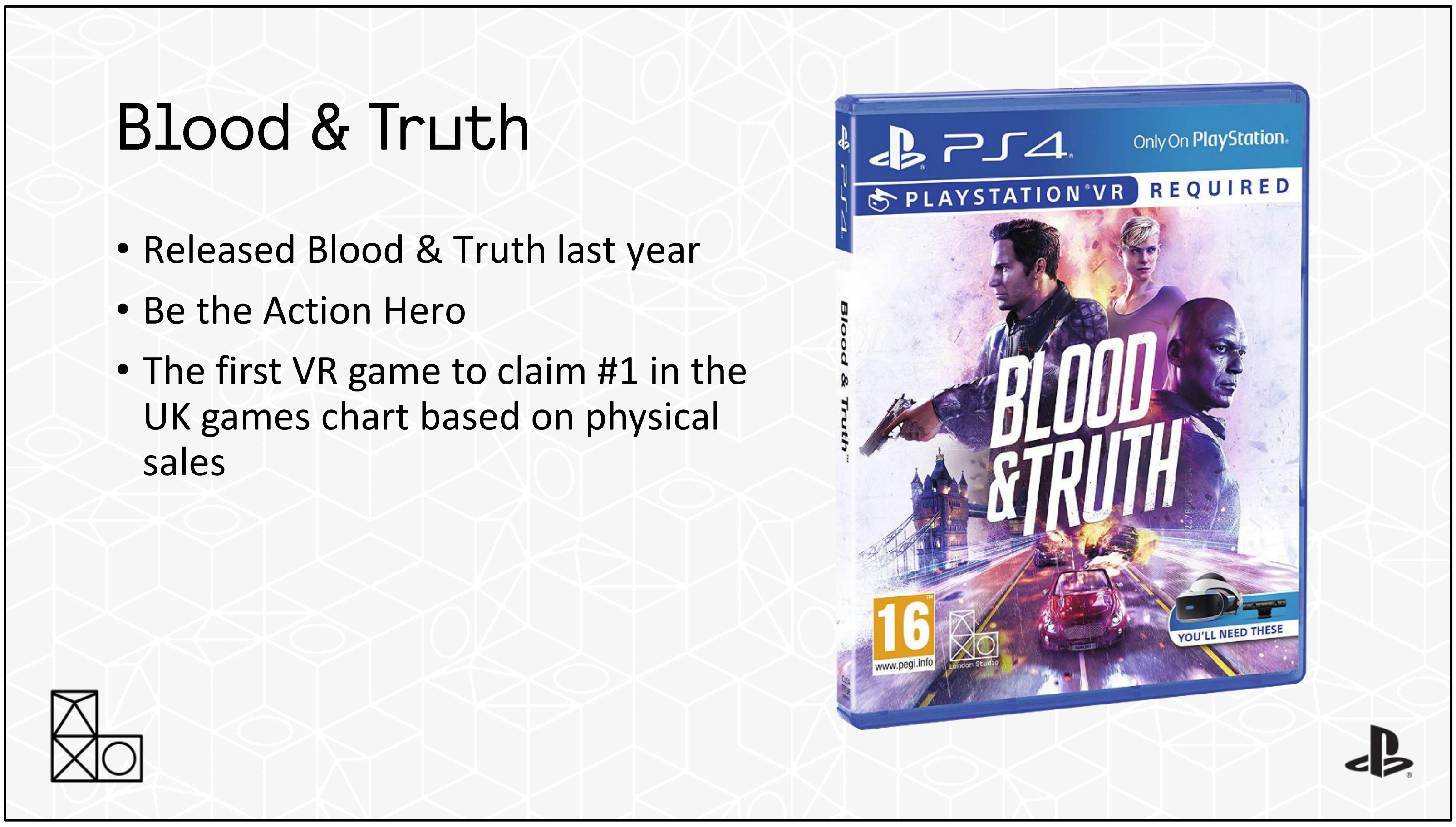
Introduction

- James Answer
- Lead Technical Artist at Sony Interactive Entertainment's London Studio.

Hello everyone. Thanks for watching my virtual SIGGRAPH talk today! I'm James Answer, and I'm Lead Technical Artist at Sony Interactive Entertainment's London Studio. We are a 1st party Sony studio right in the middle of Soho in London.

PlayStation VR Worlds

Since the earliest days of PlayStation VR we have been developing for the platform, starting with creating the range of experiences which ultimately became PlayStation VR Worlds. One of the parts of that game, The London Heist, particularly captured the imagination of players with it’s mixture of drama and real-world gunplay, so we were keen to expand the idea into a full length game.

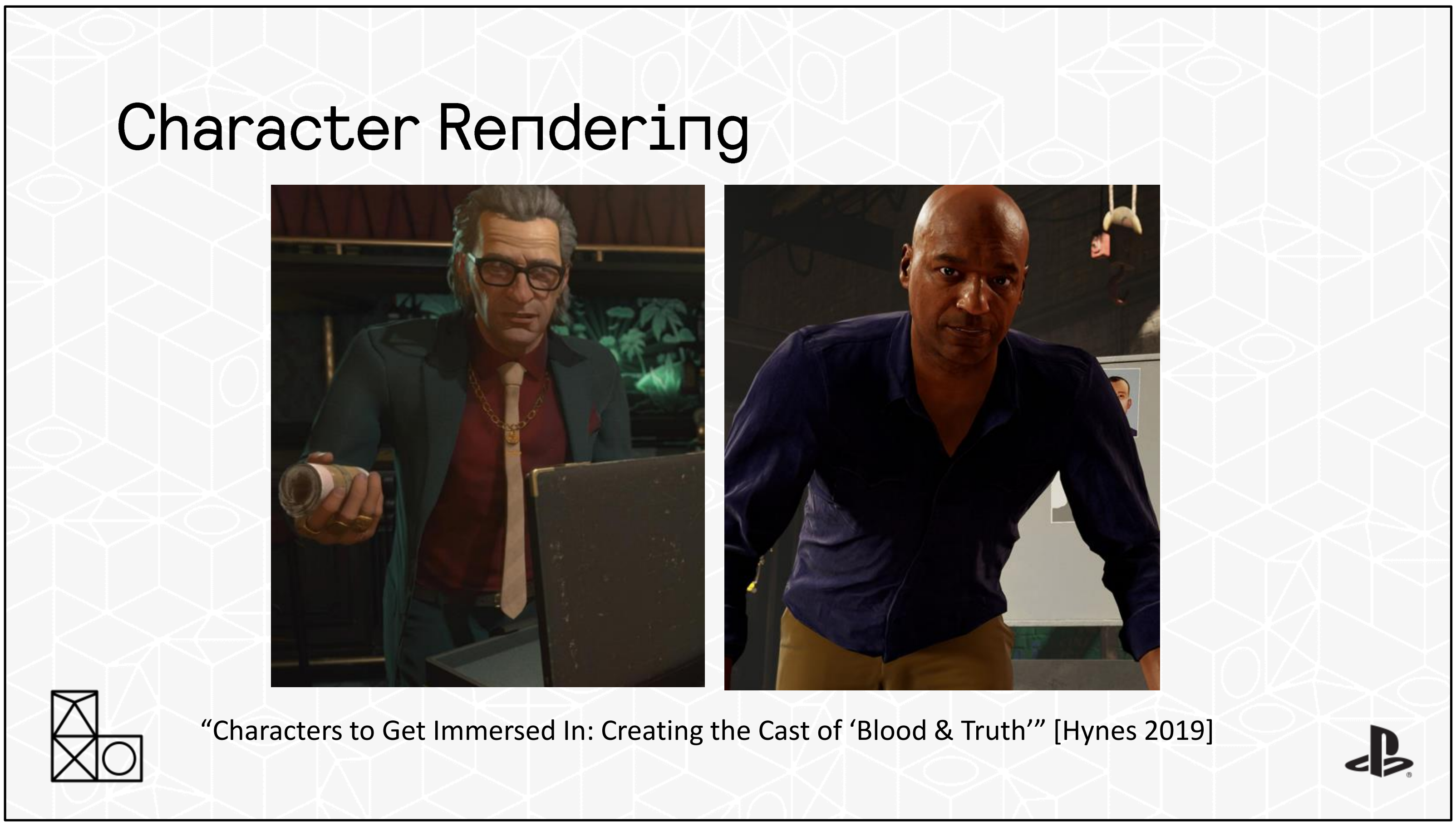


Blood & Truth

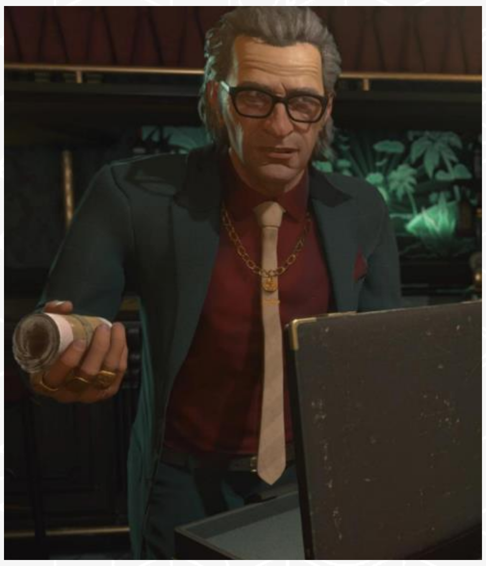
- Released Blood & Truth last year
- Be the Action Hero
- The first VR game to claim #1 in the UK games chart based on physical sales

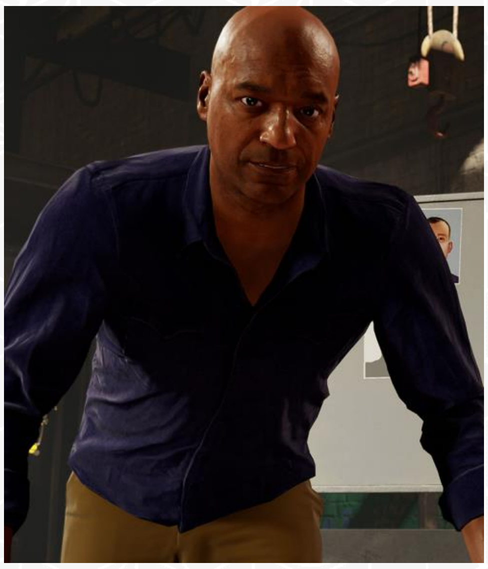
The result is Blood & Truth, which came out last year. We embraced the cinematic action of the London Heist and expanded it into a game where we wanted to use the immersivity of VR to make you feel like an action hero. It's a lot longer than London Heist, and where that game was a series of vignettes, Blood & Truth has much larger levels to move around.

We are really happy with how well Blood & Truth has been received, it was the first VR game to claim #1 in the UK games chart based on physical sales and won a number of VR game of the year awards.



Character Rendering





"Characters to Get Immersed In: Creating the Cast of 'Blood & Truth'" [Hynes 2019]

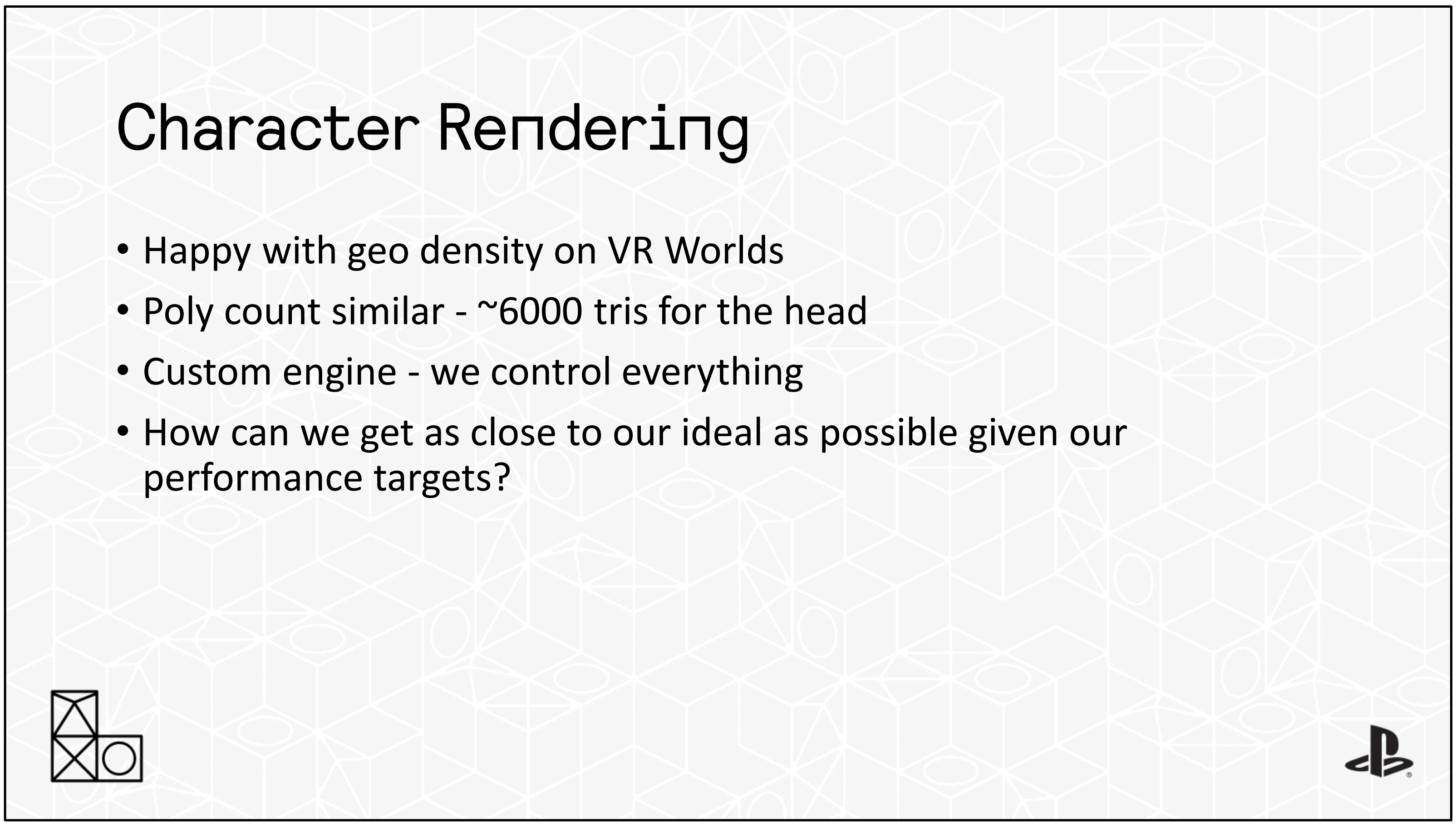
The focus of today's talk is going to be on the character rendering techniques we used.
Getting characters right was a priority for Blood and Truth – the drama sections in The London Heist were extremely important, with you spending over half of the experience interacting with characters.
The biggest change from VR Worlds was moving from stylized characters to digital doubles based on scans. We initially thought that stylized characters would read better, but after extensive testing decided that having the nuance of performance capture made it worthwhile. For more information on this process, I recommend my colleague Toby Hynes' presentation at GDC last year.
I'll show you a trailer from the game to give you an idea of the final result.

Trailer available at https://www.youtube.com/watch?v=xjwqnsOhKnI

- 60 fps at all times
- 1.4 x 1080p resolution (or higher)

There are obviously a number of games including those from our sister studios that do photorealistic characters extremely well, and we didn't want to compare unfavourably to them. But we need to hit 60 frames per second with no drops at any time, and ideally get the resolution to as close to 1.4x1080p as possible (or even higher).

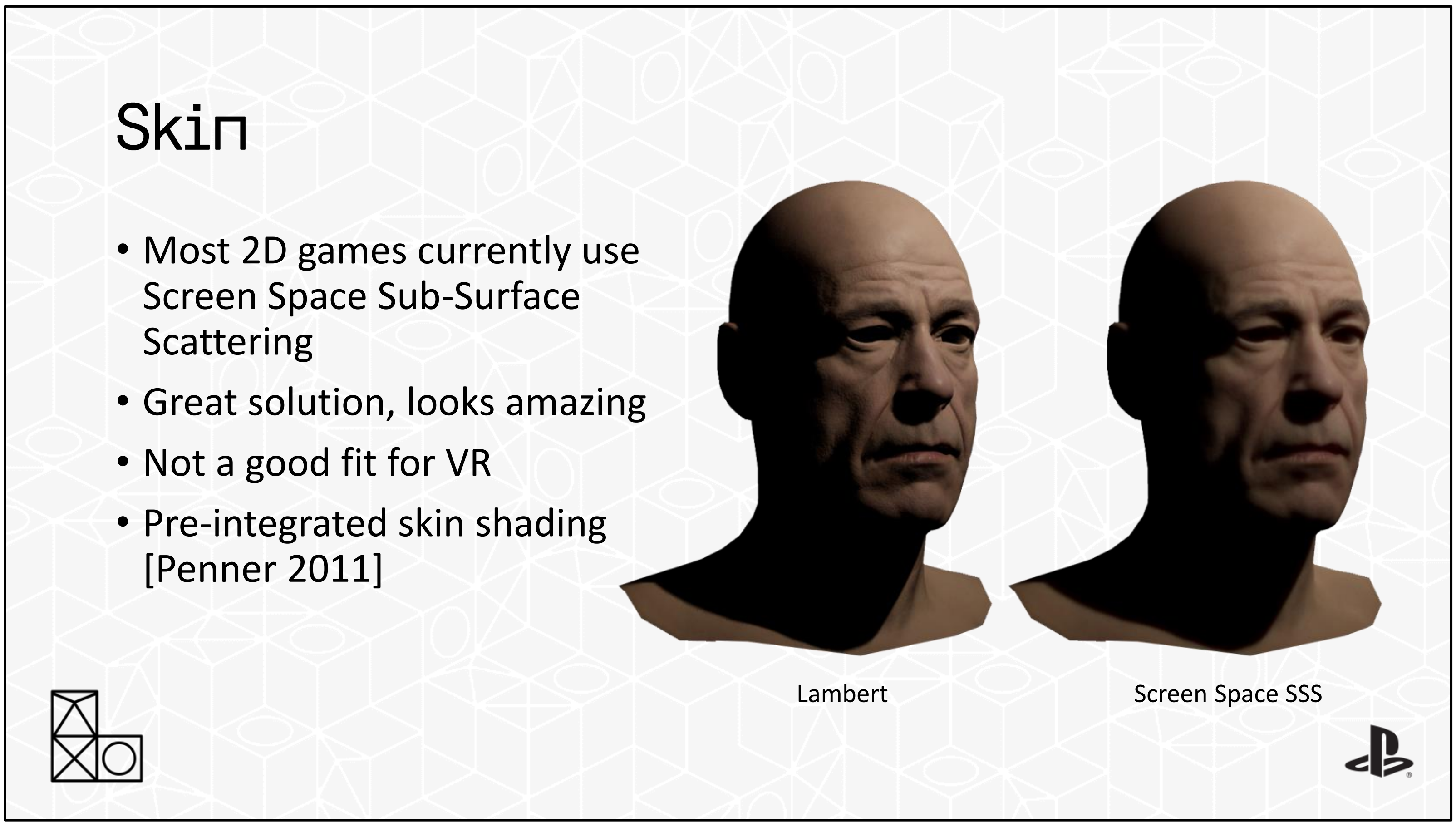
Resolution is vitally important to read eyes and expressions properly in our drama scenes. Unfortunately, some of the techniques typically used for characters in 2D games don't scale well to the pixel counts and framerate that we need for VR. Quite a few games that have rock solid 60 fps in gameplay sections drop down to 30 for their cutscenes, largely to accommodate these expensive techniques. This isn't an option for us, so we needed to find more efficient alternatives to maintain both performance and the quality we wanted.



Character Rendering

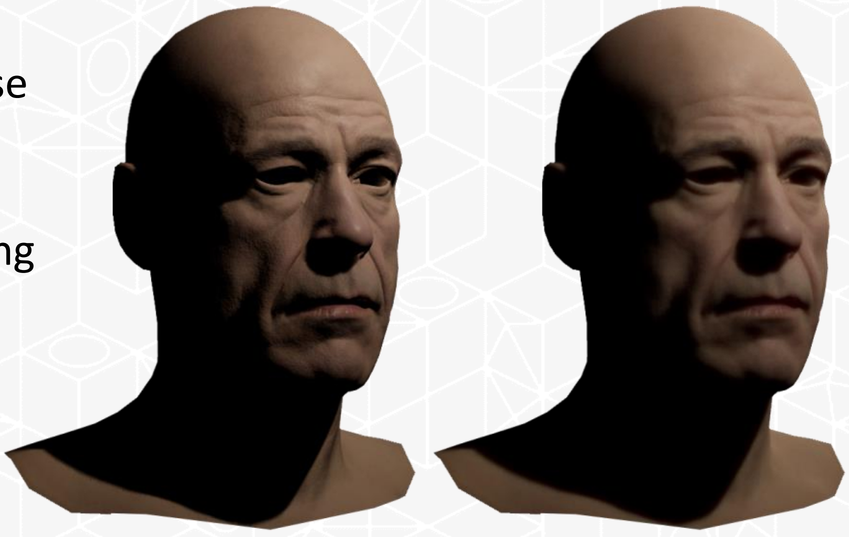
- Happy with geo density on VR Worlds
- Poly count similar - ~6000 tris for the head
- Custom engine - we control everything
- How can we get as close to our ideal as possible given our performance targets?

We were happy with the poly count on VR Worlds, the heads were about 6000 triangles and were confident that this was a good level of detail at the distances you typically see characters at – there are no close ups on faces to worry about. But we needed to work out how we could replicate the character rendering effects people expect. We use a custom engine at London Studio with a forward renderer, which allows us a great deal of flexibility for adding custom shading models and techniques, so the question was how can we use this to get as close to our ideal as possible given our performance targets?



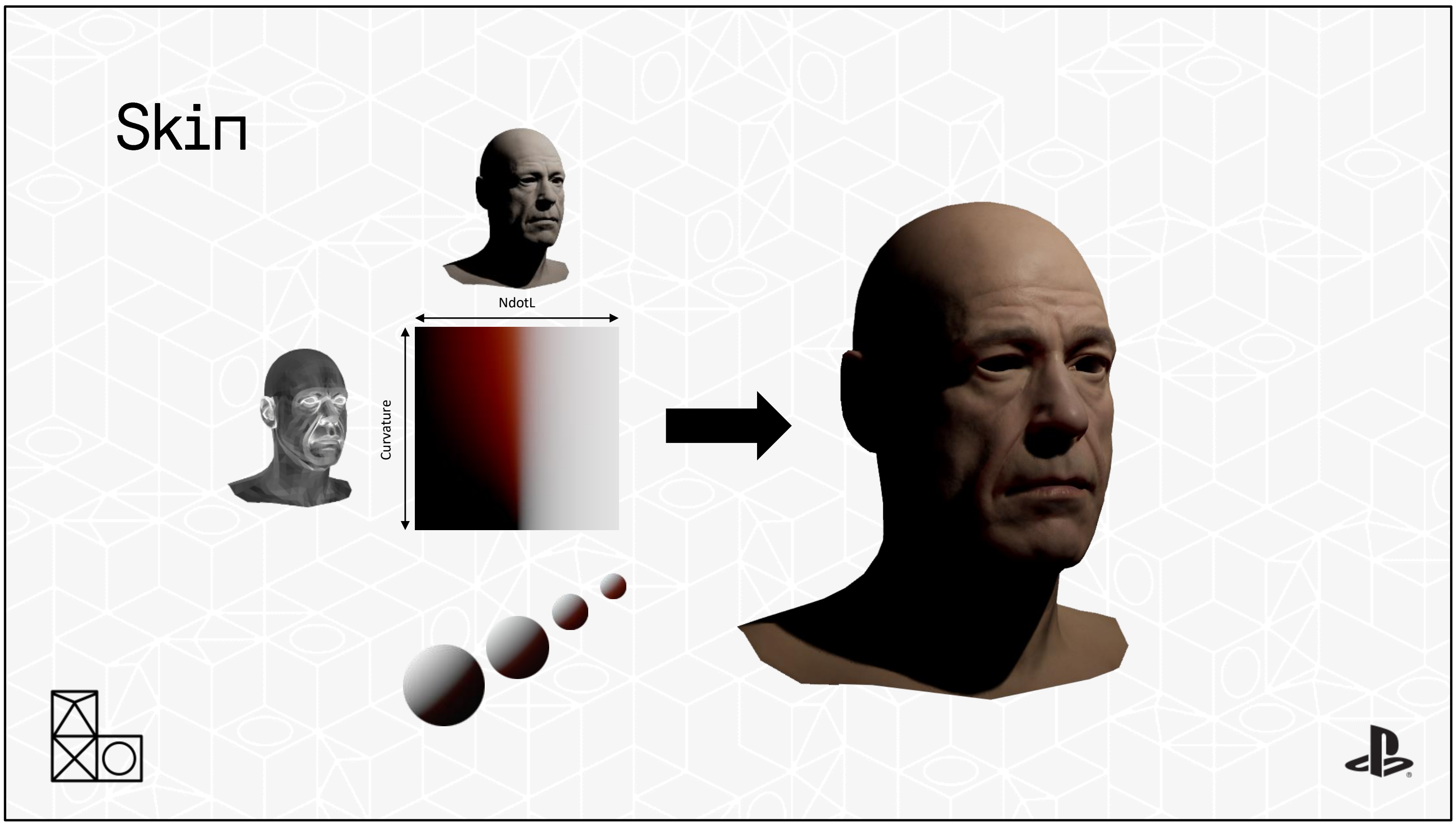
Skin

- Most 2D games currently use Screen Space Sub-Surface Scattering
- Great solution, looks amazing
- Not a good fit for VR
- Pre-integrated skin shading [Penner 2011]



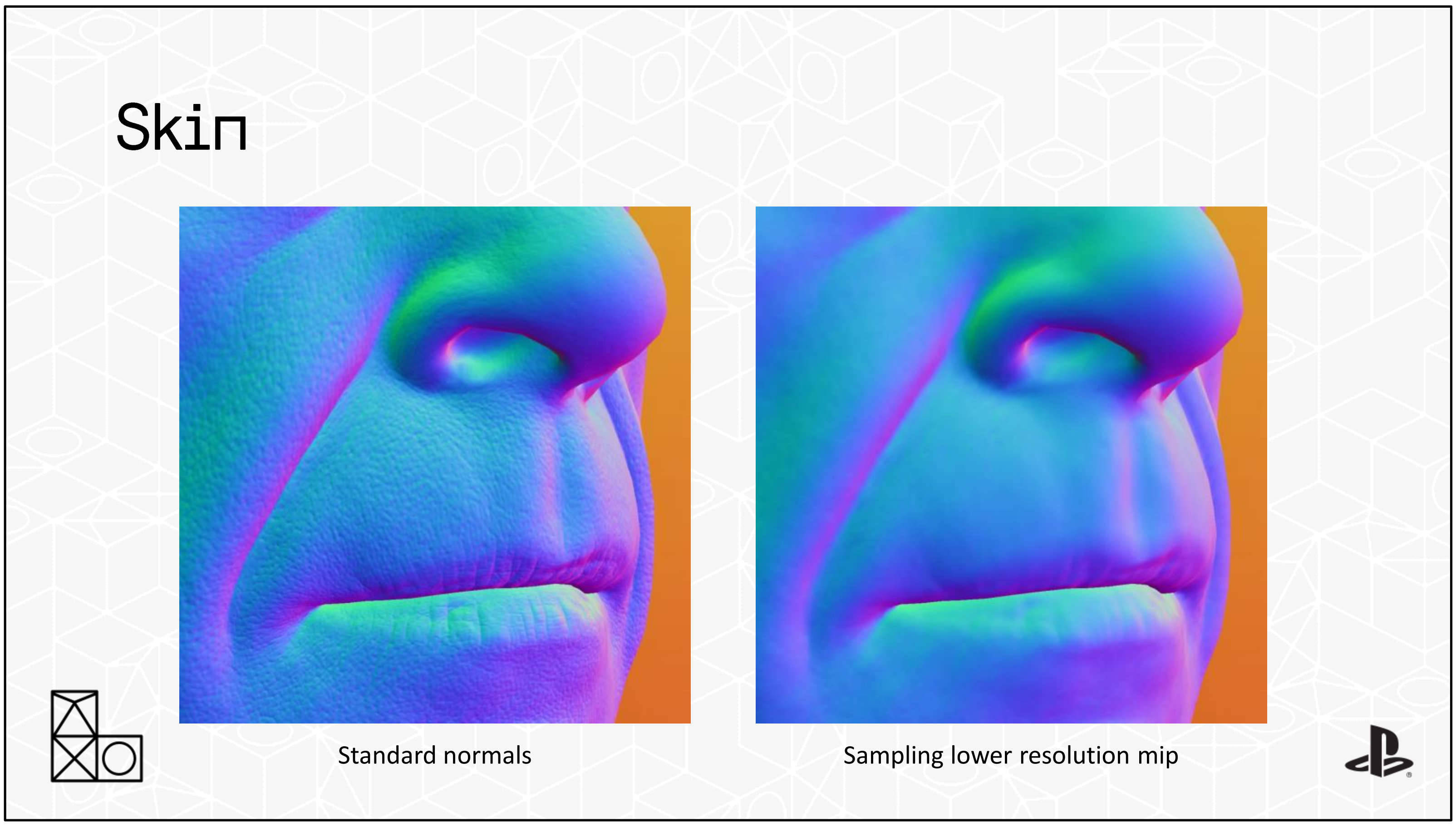
Lambert Screen Space SSS

I guess the first and most obvious component of character rendering is skin shading. It’s vital to simulate the effects of scattering in the skin, as otherwise you get an orange peel look. Most 2D games these days use screen space sub-surface scattering, which is a great solution.
However, it often doesn’t work well with VR - it can be expensive at high resolutions, and in general we tried to keep all the lighting into a single pass to reduce memory bandwidth costs.
Luckily there is a well established alternative in the form of Eric Penner’s pre-integrated skin shading.

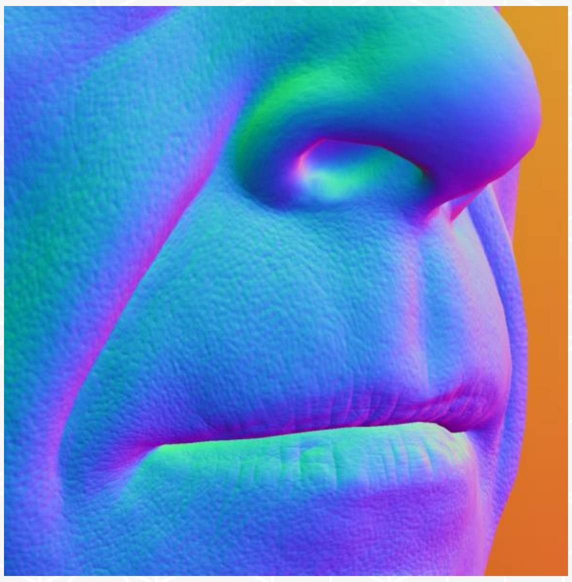


Skin

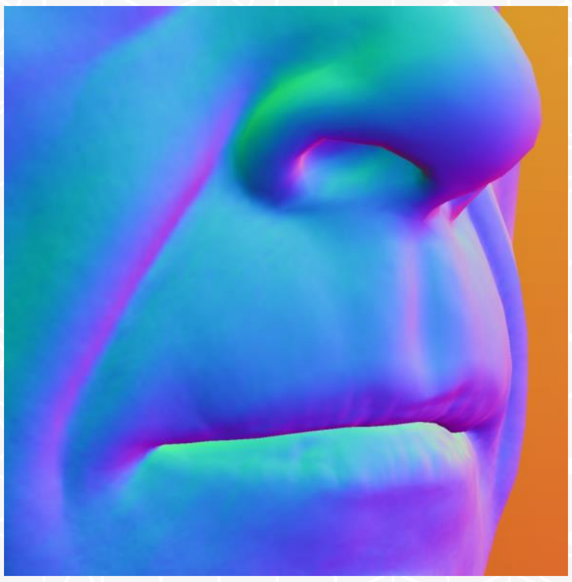
<click>Pre-integrated skin shading uses the observation that as the curvature of the mesh increases, light scatters further through it. The effects of subsurface scattering can be approximated by baking the scattering on spheres of different radii to a lookup texture
<click>We can then use the curvature of the mesh (essentially the radius) and NdotL term of the lighting to choose the appropriate lookup coordinate. You can generate the curvature by using derivatives of the normal and position in the fragment shader. This updates as the face moves, but has a faceted look. Or you can simply bake the curvature of the mesh offline into vertex colours, at the cost of it not being dynamic. This the approach we used on Blood & Truth.
<click>The result captures the main features we want to see in a skin shader, with soft diffuse lighting and colour bleeding.



Skin



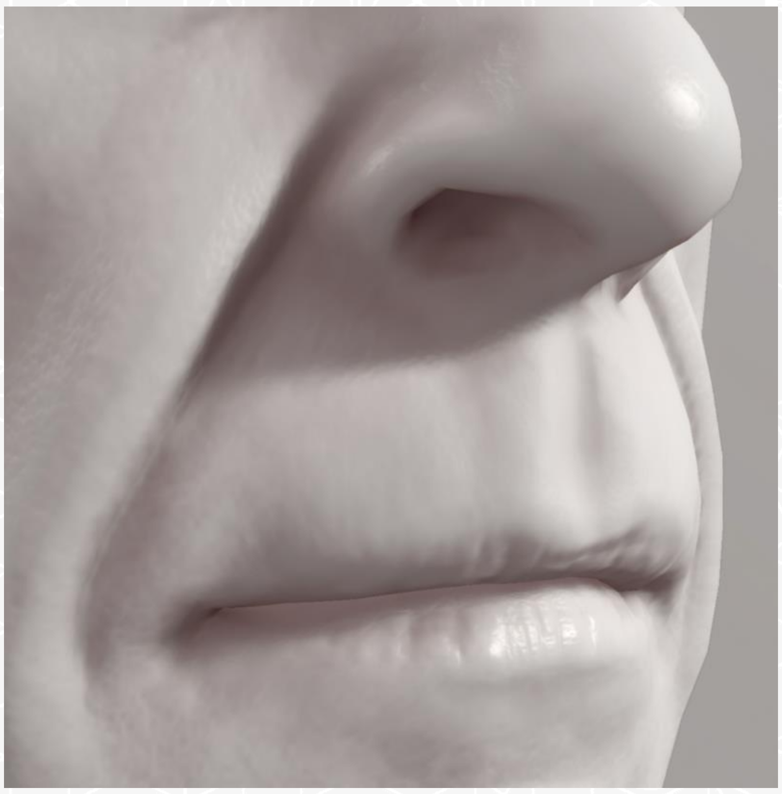
Standard normals



Sampling lower resolution mip

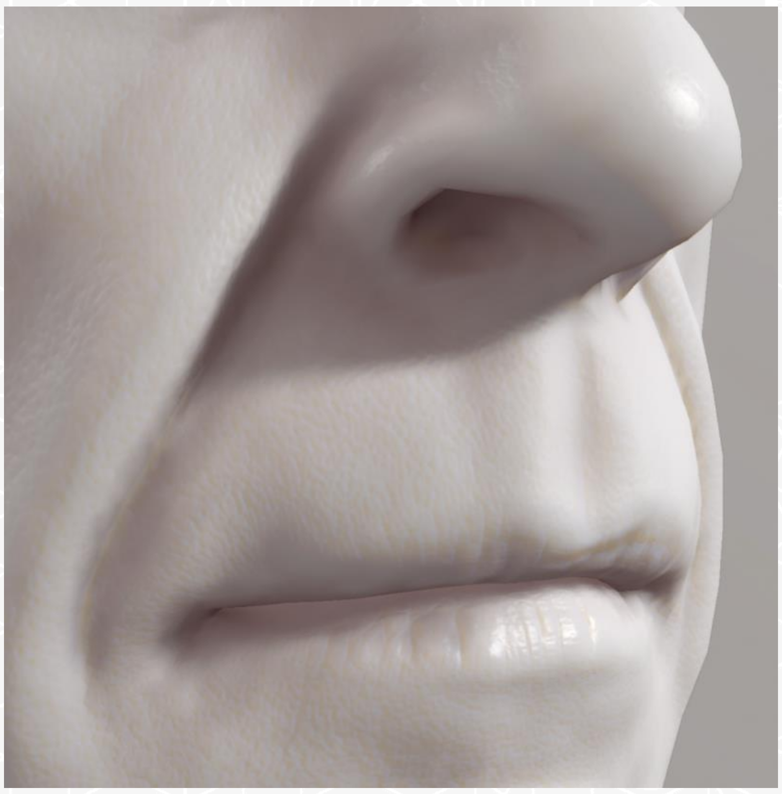
The other main component to this technique is that diffuse light can scatter through small scale bumps, which is simulated by using a blurred normal. This can be cheaply approximated by sampling the normal map again at a lower mip. The standard normal map is used for specular lighting, and the blurred normals can be used for the diffuse.

Skin

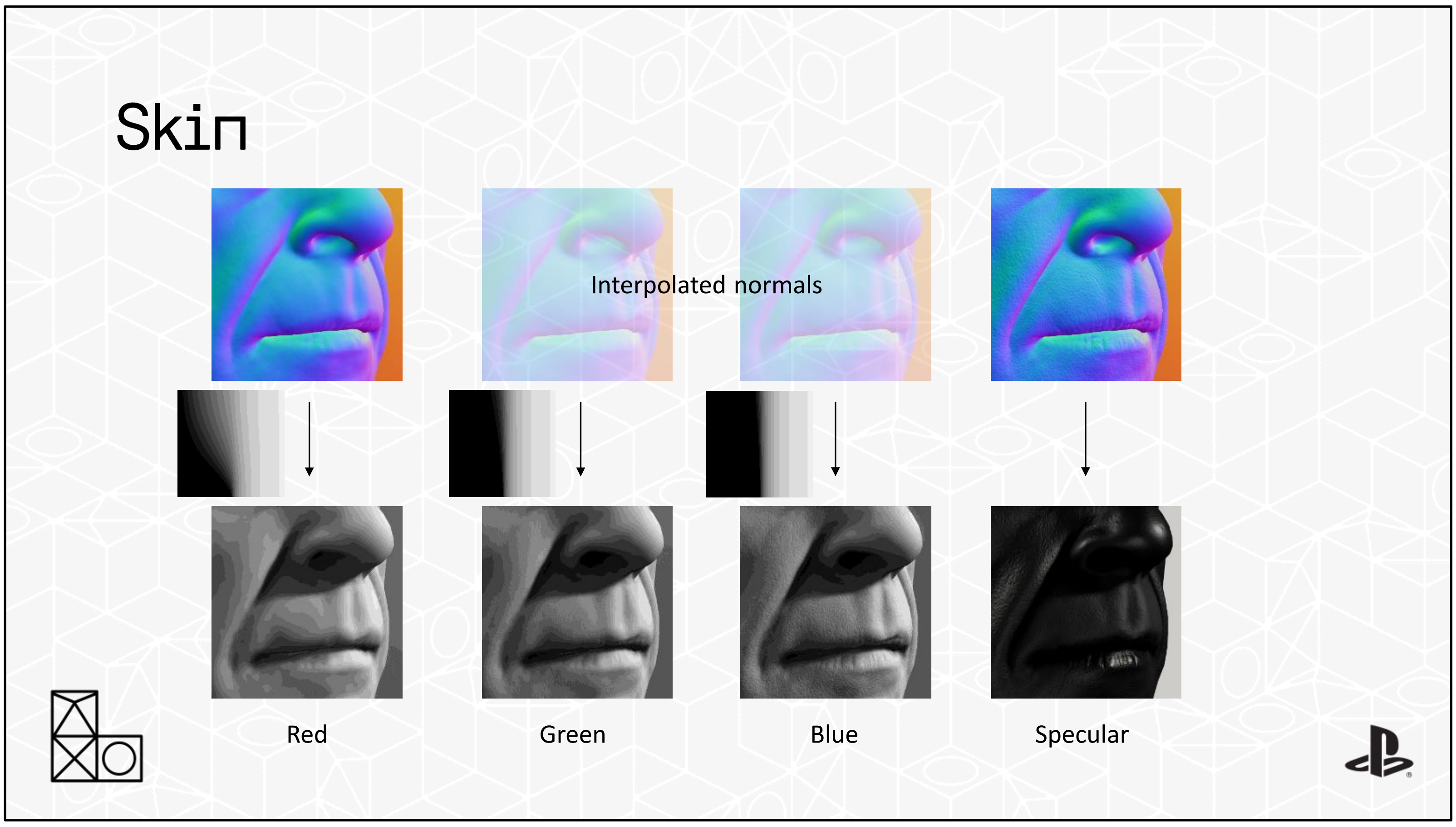


For VR Worlds we only used the blurred normal for diffuse lighting, and I've seen other versions of this technique do the same thing. This makes the diffuse lighting too soft and the temptation is to use a less blurred normal to compensate.

Skin



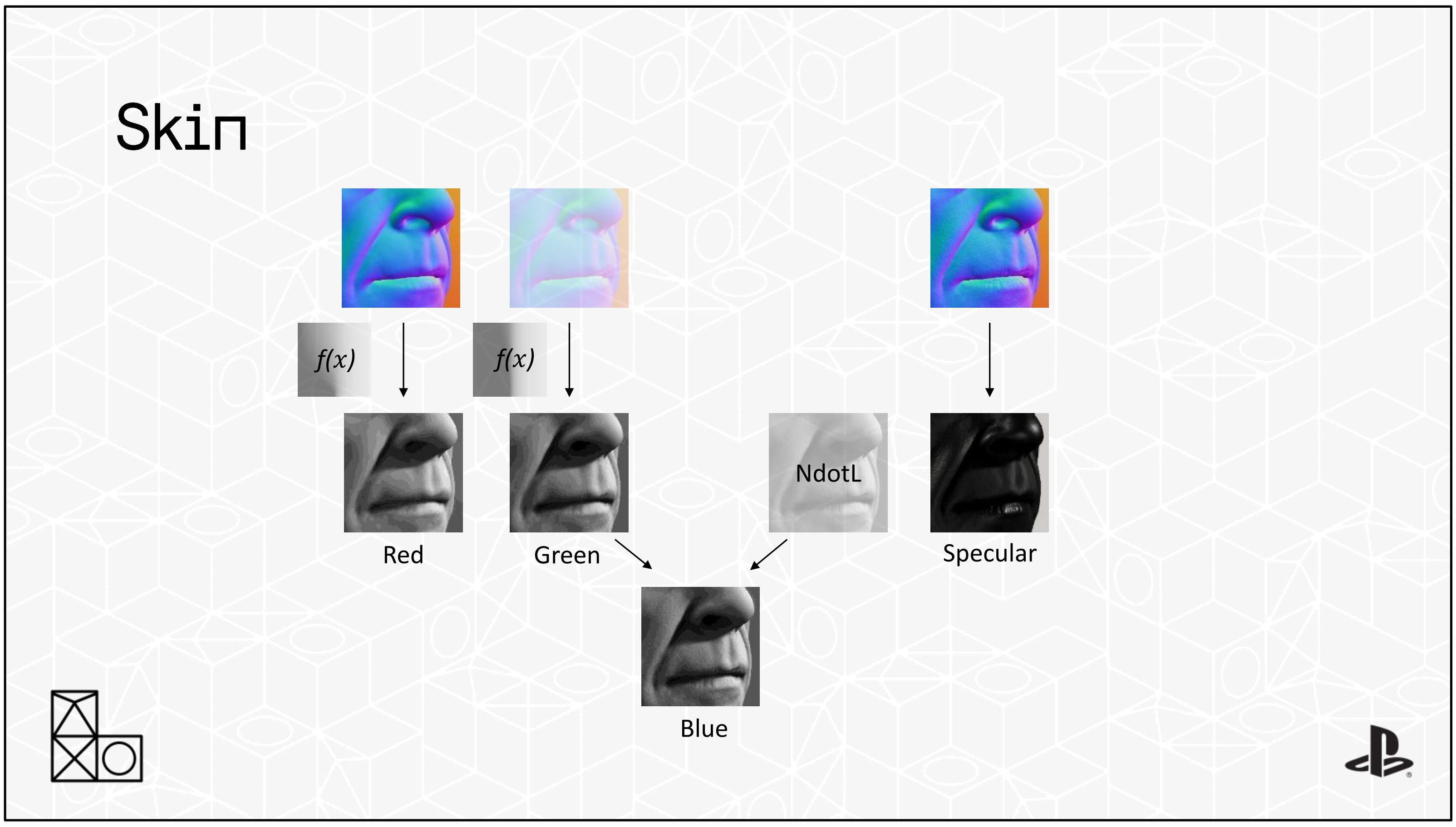
Penner’s paper recommends using different normals for the red green and blue colour components, which gives us a sharper result whilst still scattering realistically.



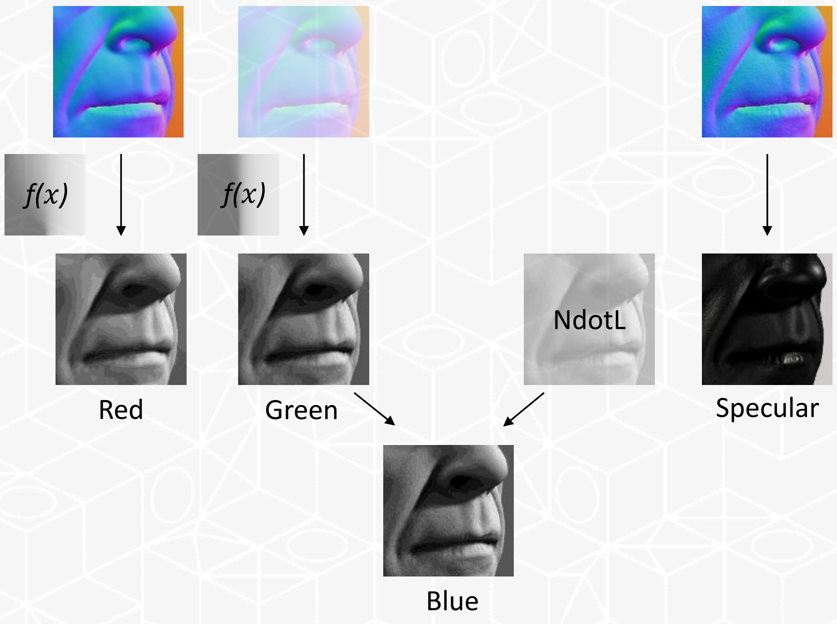
Skin

You can do this by using the blurred normal for the red channel, the regular normal for the specular, and a mixture of the two for the green and blue channels.

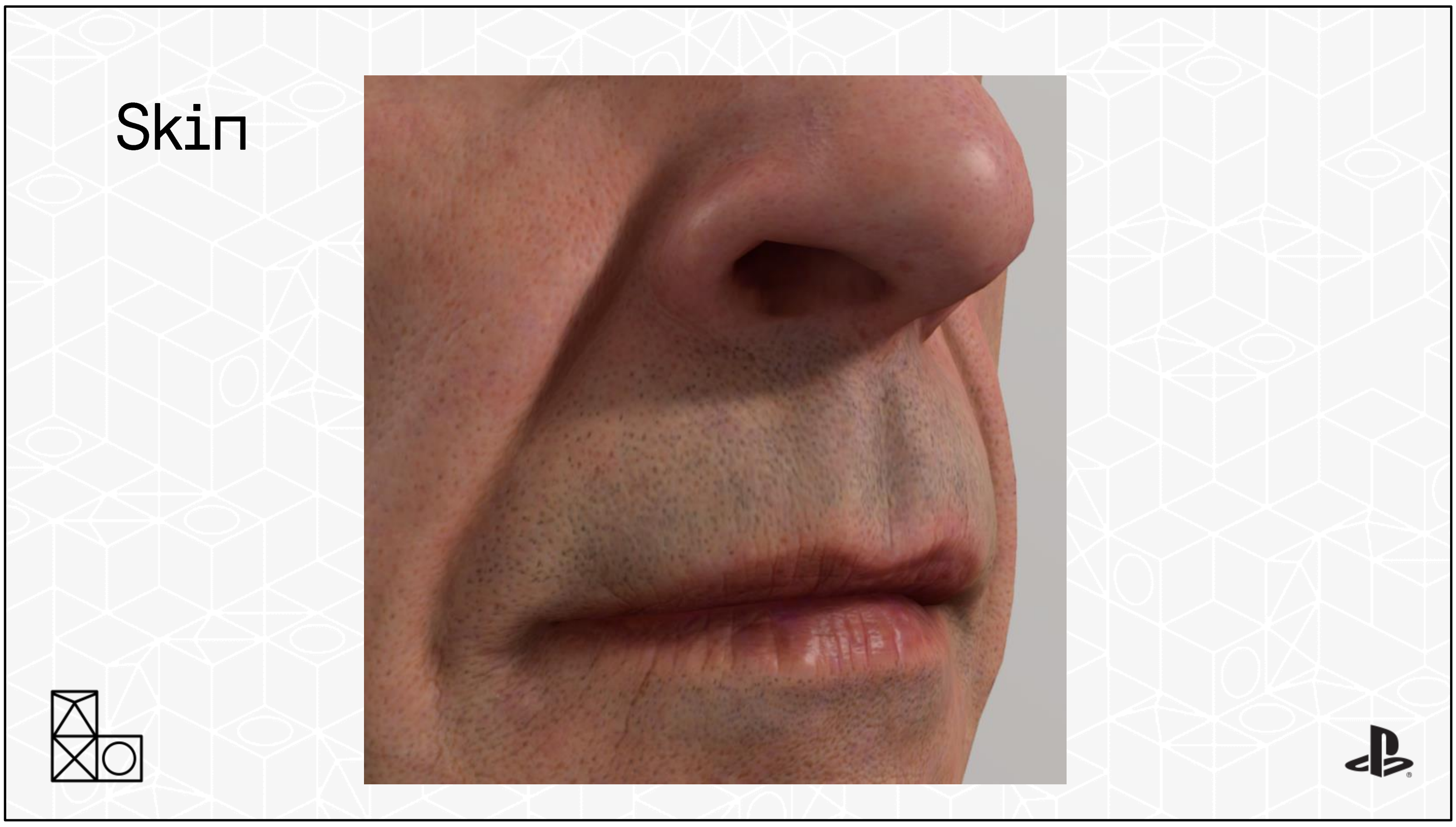
This does mean having 3 reads of the lookup texture though, so we made a couple of approximations.



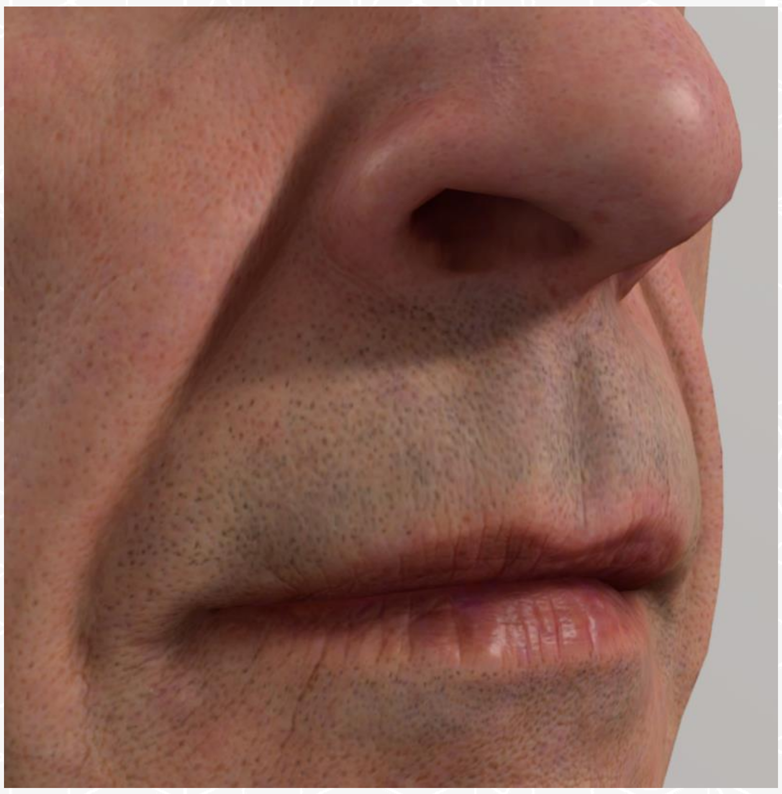
Skin



The first was to use a fitted curve approximation to the lookup texture. As a further optimization, we noticed that blue channel result was very close to lambert, so we use a simple lerp between NdotL and the result of the green channel.



Skin



This combined gives us a plausible result for skin, and can all be calculated in a single render pass without having the bandwidth cost of having to store out diffuse and specular lighting results or running a blur pass. We do lose the ability to blur the shadows, and sharp features on the face will not scatter quite correctly, but overall this technique has worked very well for us.

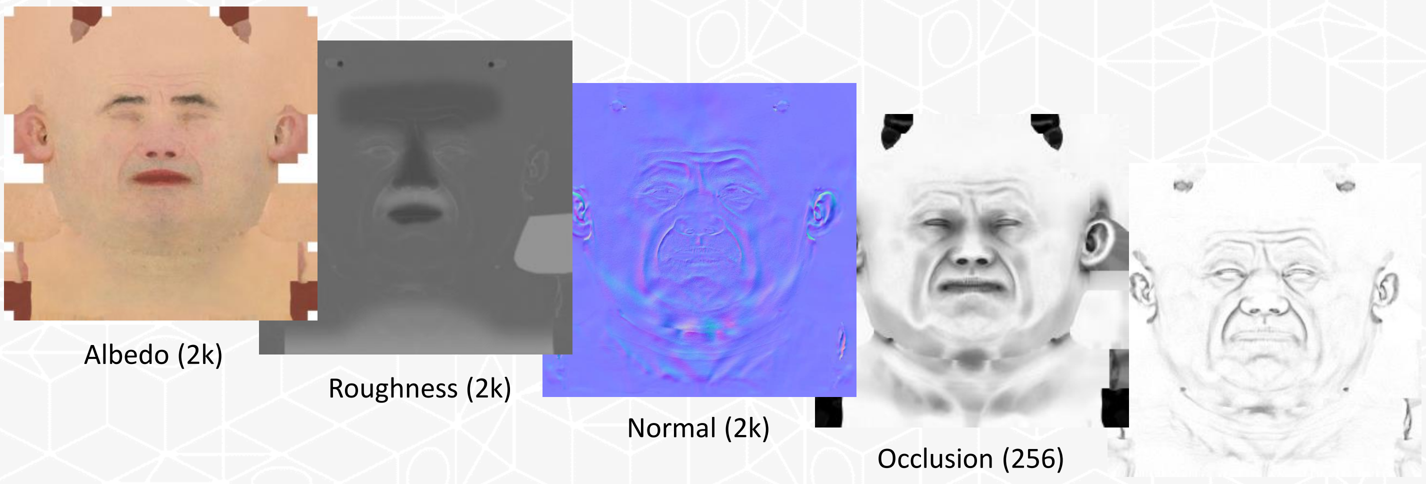
(Note: Penner's paper does give some ideas for dealing with shadows, but as we used a deferred shadow buffer with the attenuation already applied, we couldn't take advantage of this.)

Textures

- Highly detailed scans
- 3 wrinkle maps for each character
- Not just normals, also albedo, occlusion, bent normals, cavity map

Next is textures. We partnered with 3Lateral on the character scanning, and we got back highly detailed 4k textures and 3 sets of wrinkle maps including albedo changes. Because the poly count on the face was relatively low, we wanted to extract as much information out of the bakes as possible, so we got occlusion and bent normal bakes for not only the base pose, but also the wrinkle maps.

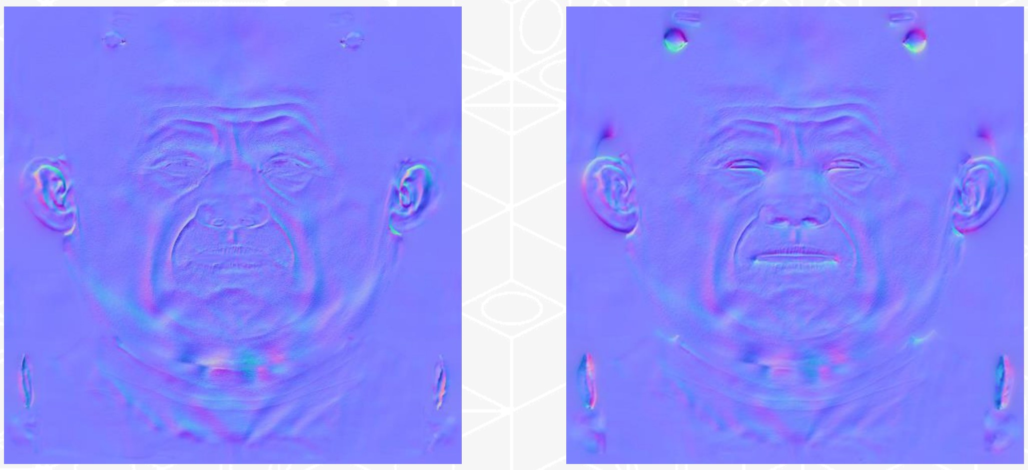
Textures



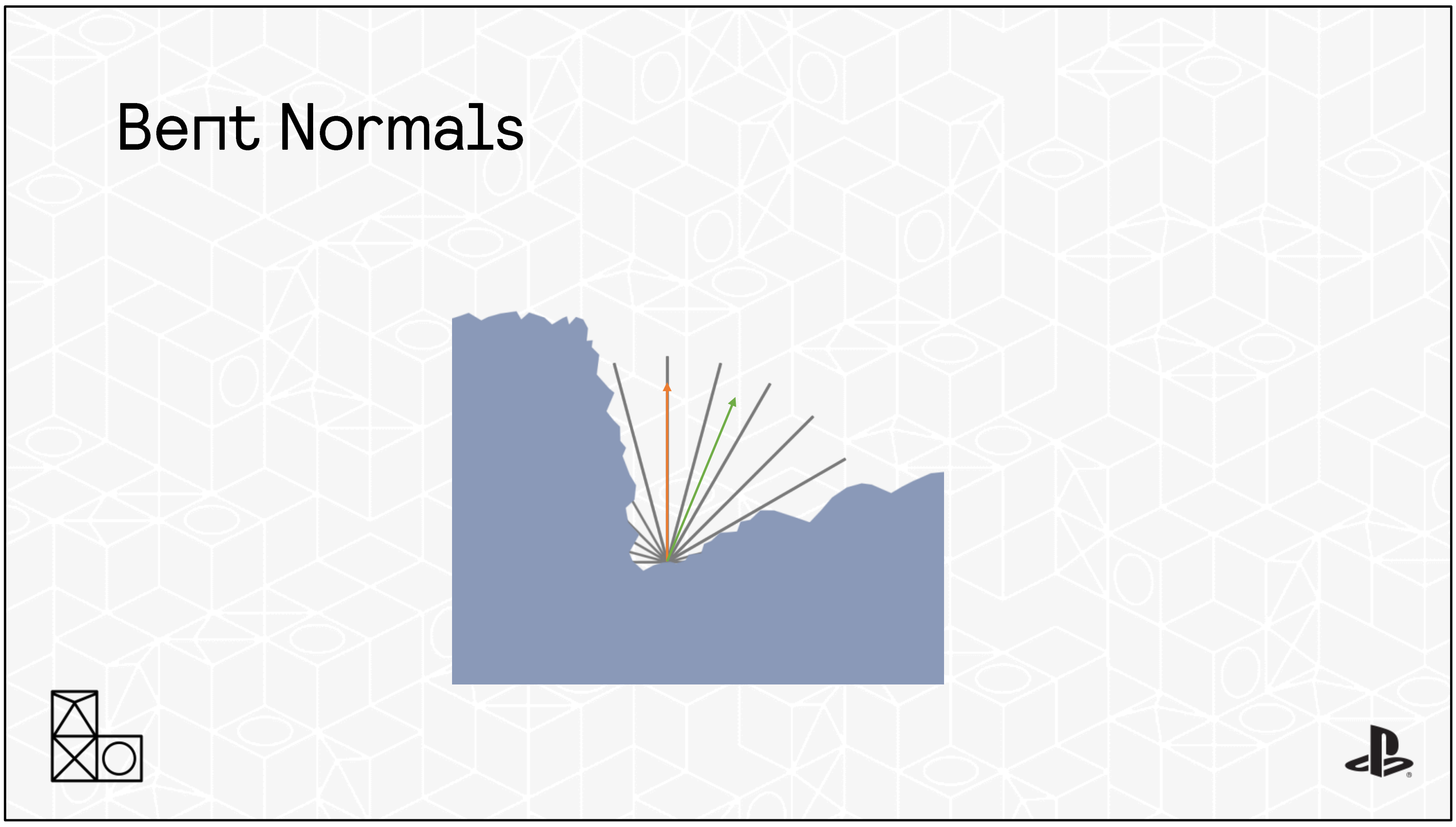
- Roughness modified by high resolution normals using specular AA filter [Neubelt & Pettineo 2013]

These are the core textures used on the skin shader. Although we have 4k maps available, the highest resolution maps we use at runtime are 2k. But with an efficient UV layout, this is still enough to capture pore detail. We do take advantage of the extra resolution of the normal map by converting the high frequency normals lost on downsampling to roughness, using the specular anti-aliasing filter technique presented by Ready at Dawn at Siggraph 2013, and here implemented as a pixel processor in Substance. This takes the variance of the normal across the footprint of the downsampled pixel and modifies the roughness value, giving us additional microsurface information.
The occlusion map is 256x256, partially because it is low frequency detail anyway, but also because using a highly detailed map would re-introduce lighting information in the diffuse that we are trying to blur out. We do store an additional cavity map at a higher resolution though, to mask specular reflection in shadowed areas and pores.

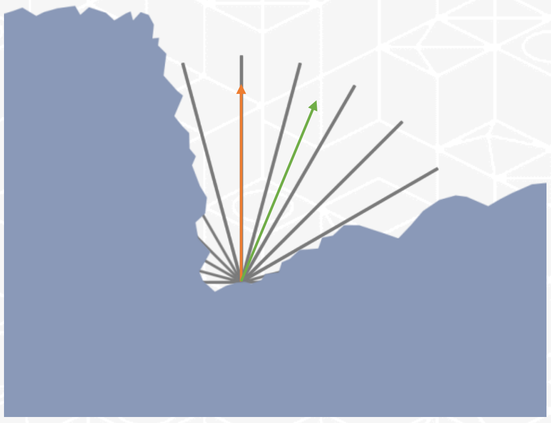
Bent Normals



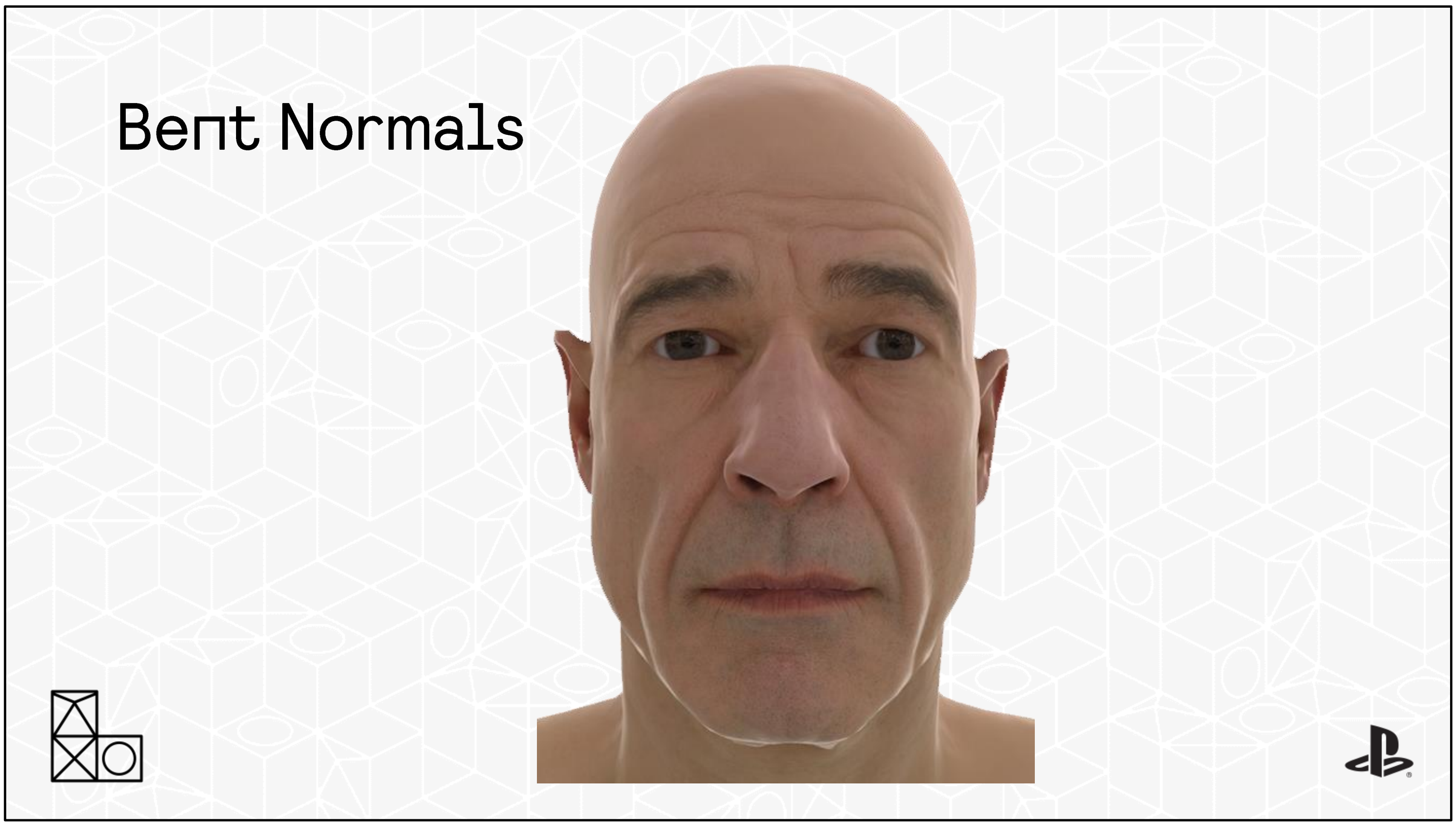
I mentioned earlier that we baked bent normal maps. This is an example of one compared to the normal map, but what does it represent?



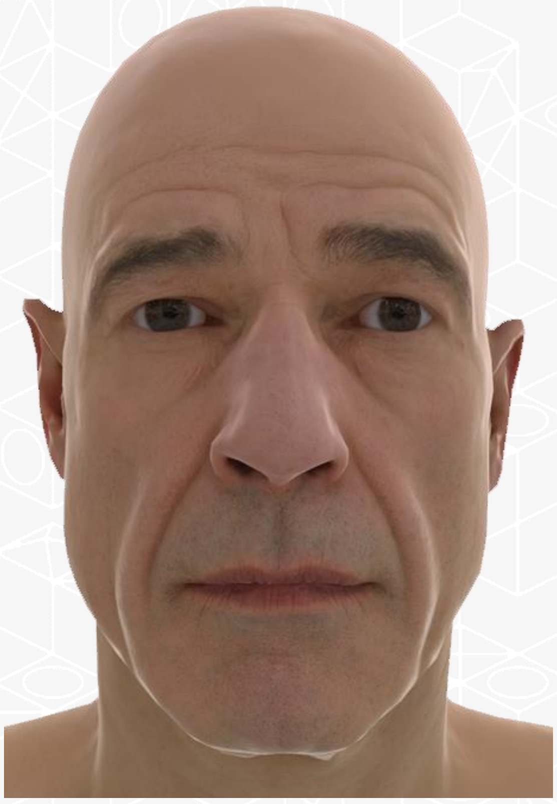
Bent Normals



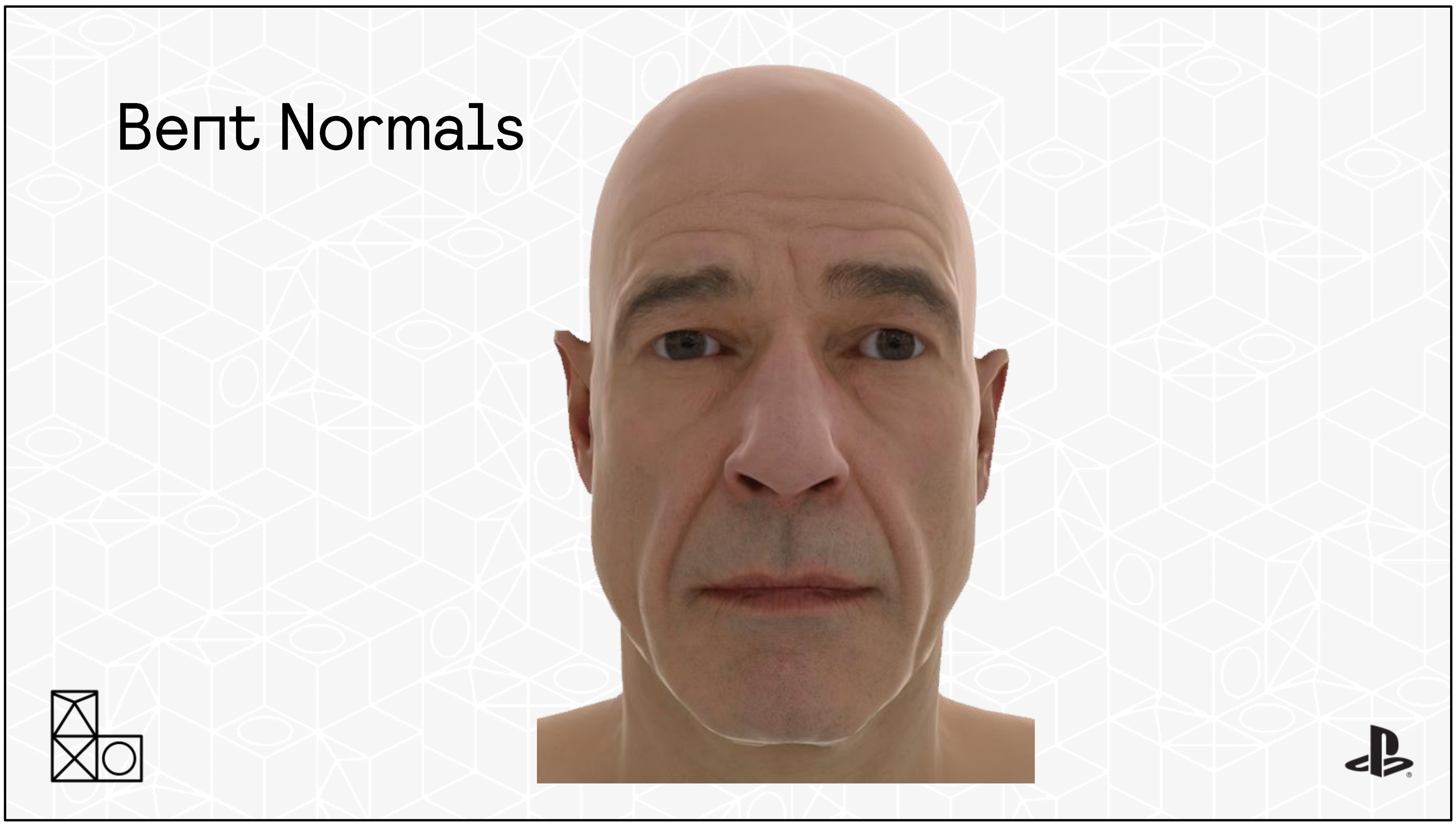
A bent normal is the cosine weighted average unoccluded direction<click>
When you generate ambient occlusion, you fire out rays from a point in a hemisphere <click>
Then work out how occluded that point is by how many of them hit a surface.<click>
The bent normal is the average of all the rays which don't hit anything. And what this gives us is a better direction to fetch ambient light from.



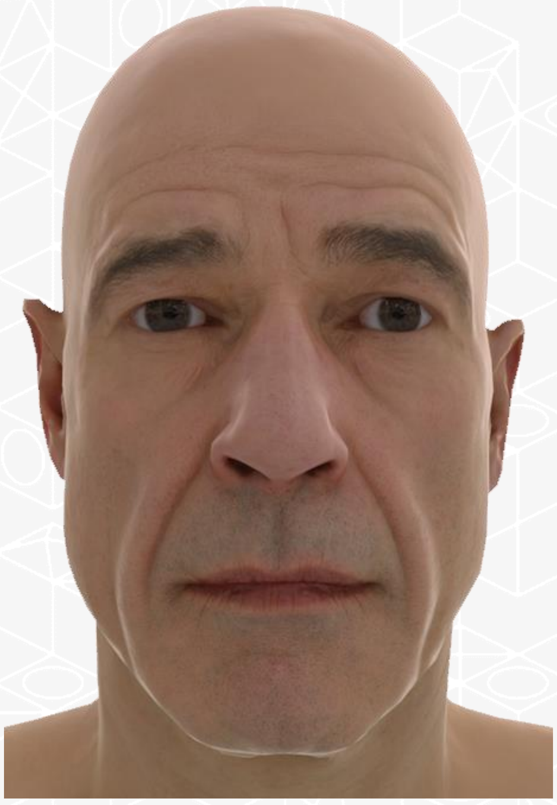
Bent Normals



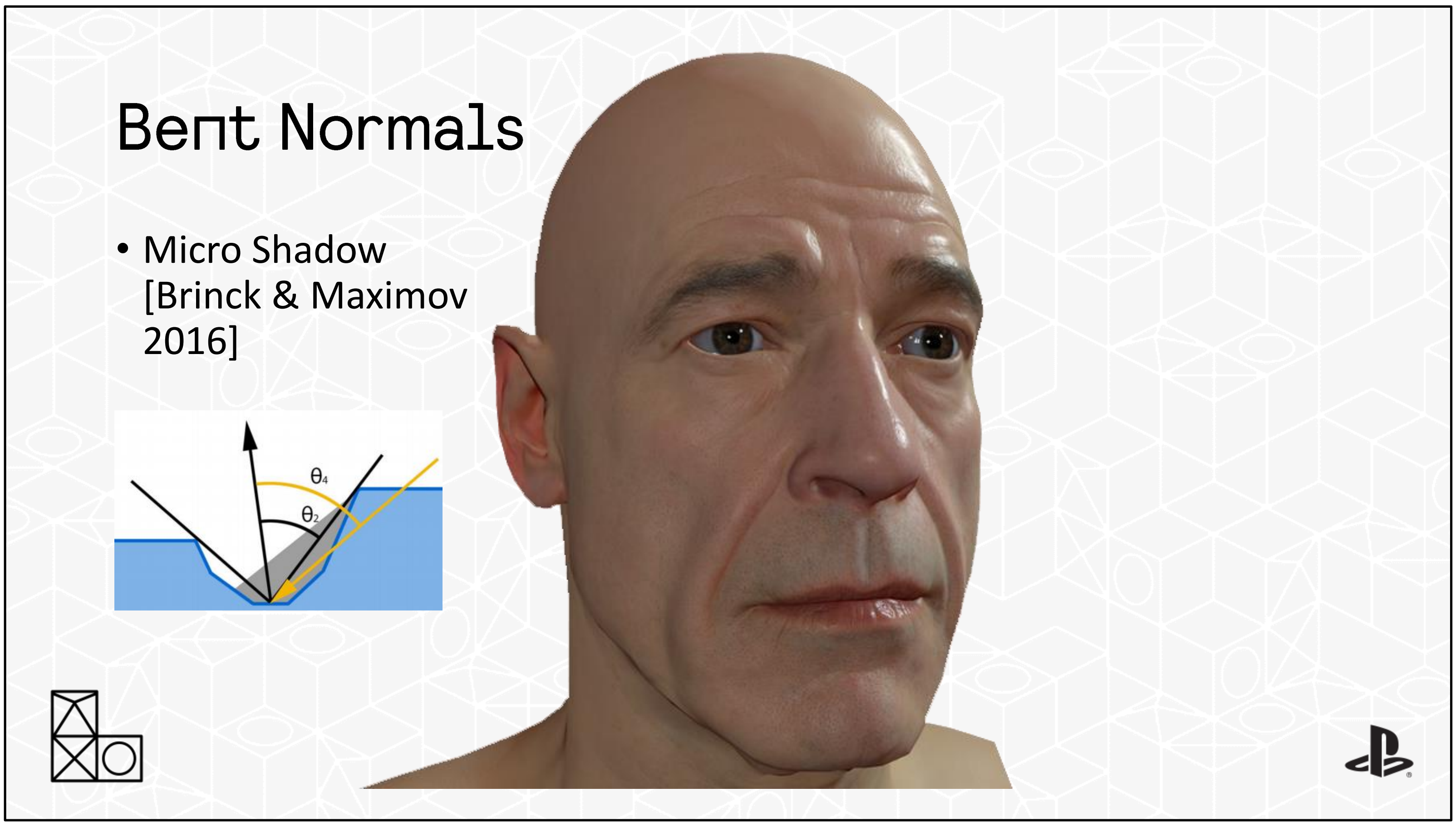
This is a shot using only ambient lighting with standard normals. You can see here we have some leaking around the lips and sides of the nose.



Bent Normals

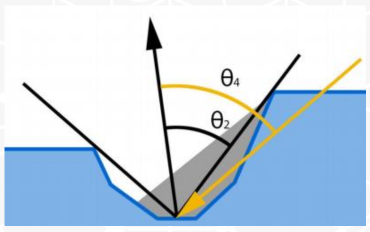


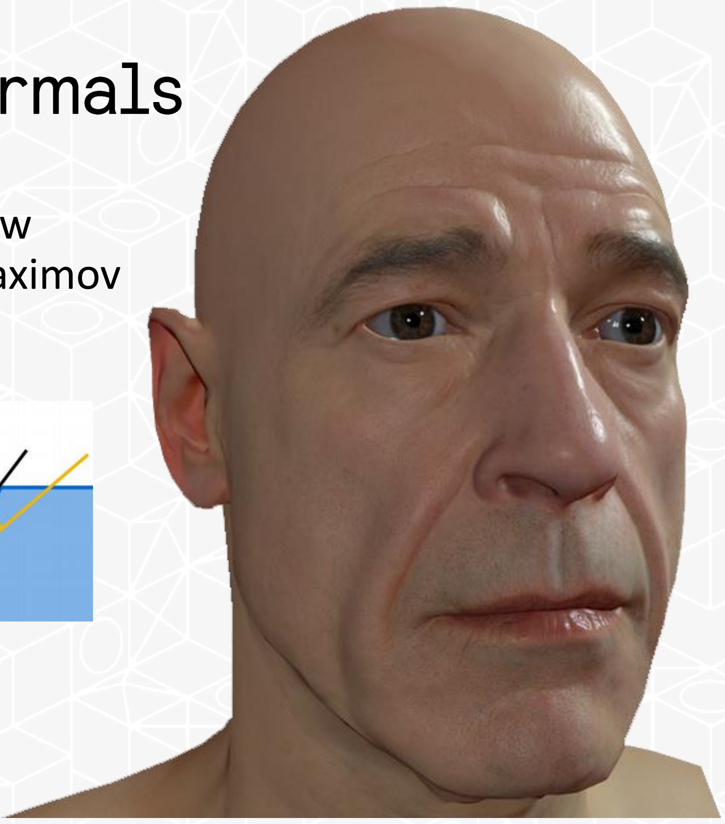
Compared to using bent normals, with which we get reduced light leaking



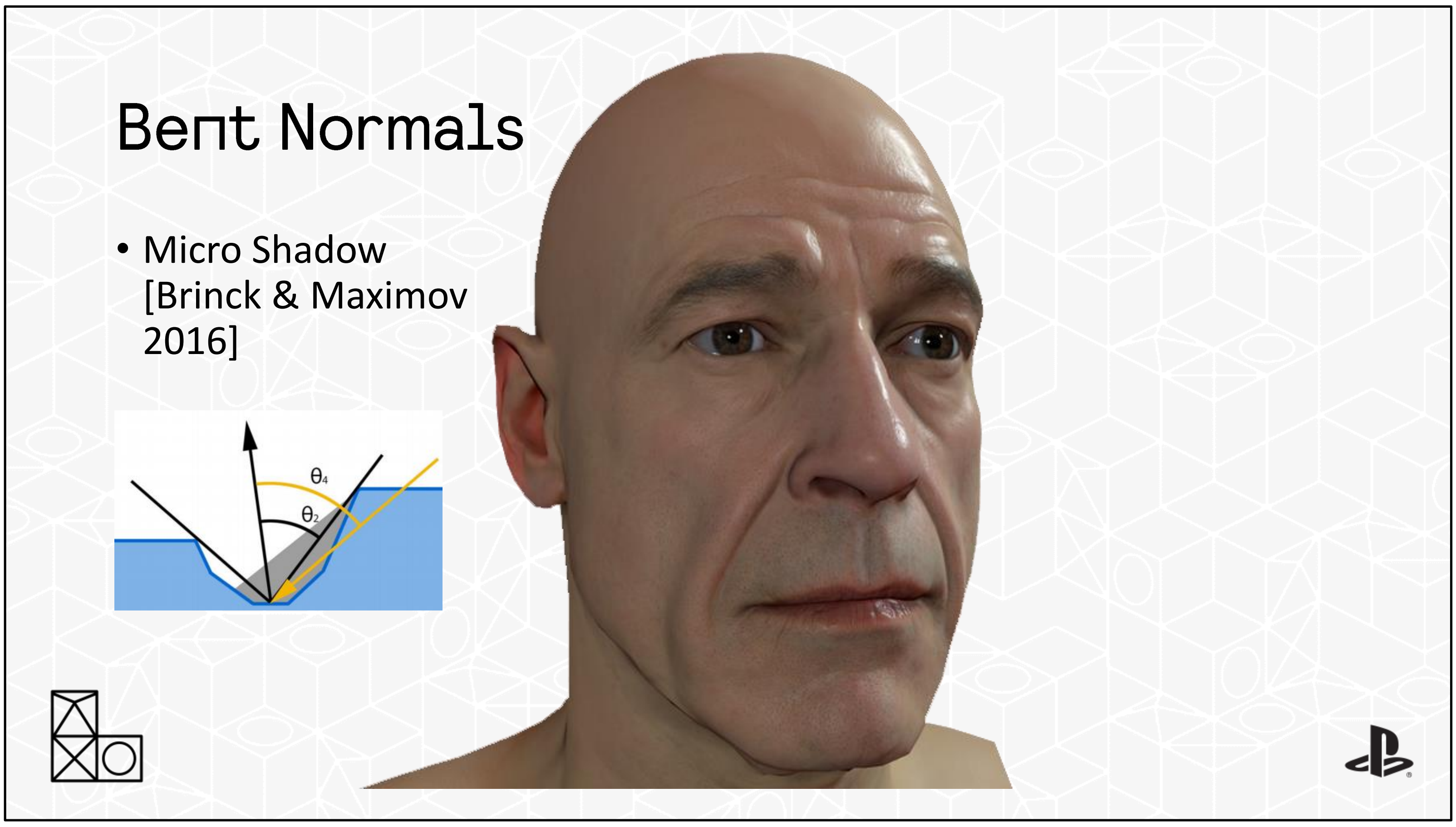
Bent Normals

- Micro Shadow [Brinck & Maximov 2016]



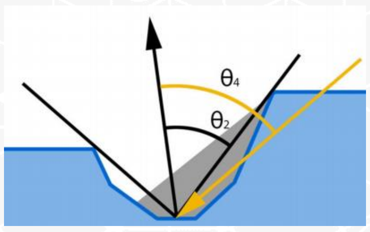


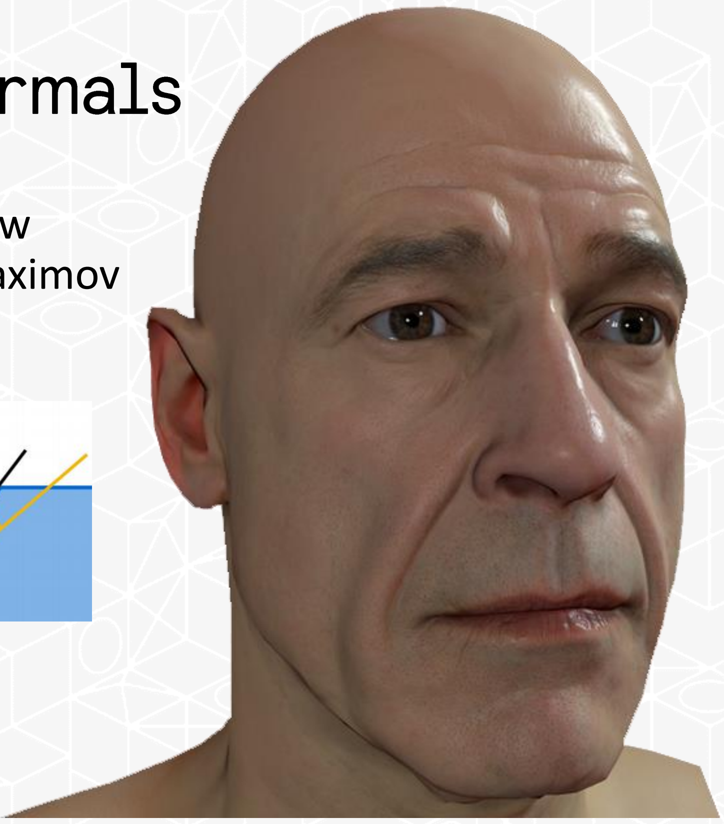
We can also use the bent normals to approximate shadows on direct lights, based on the micro shadow technique presented by my colleagues Waylon Brinck and Andrew Maximov at Naughty Dog.



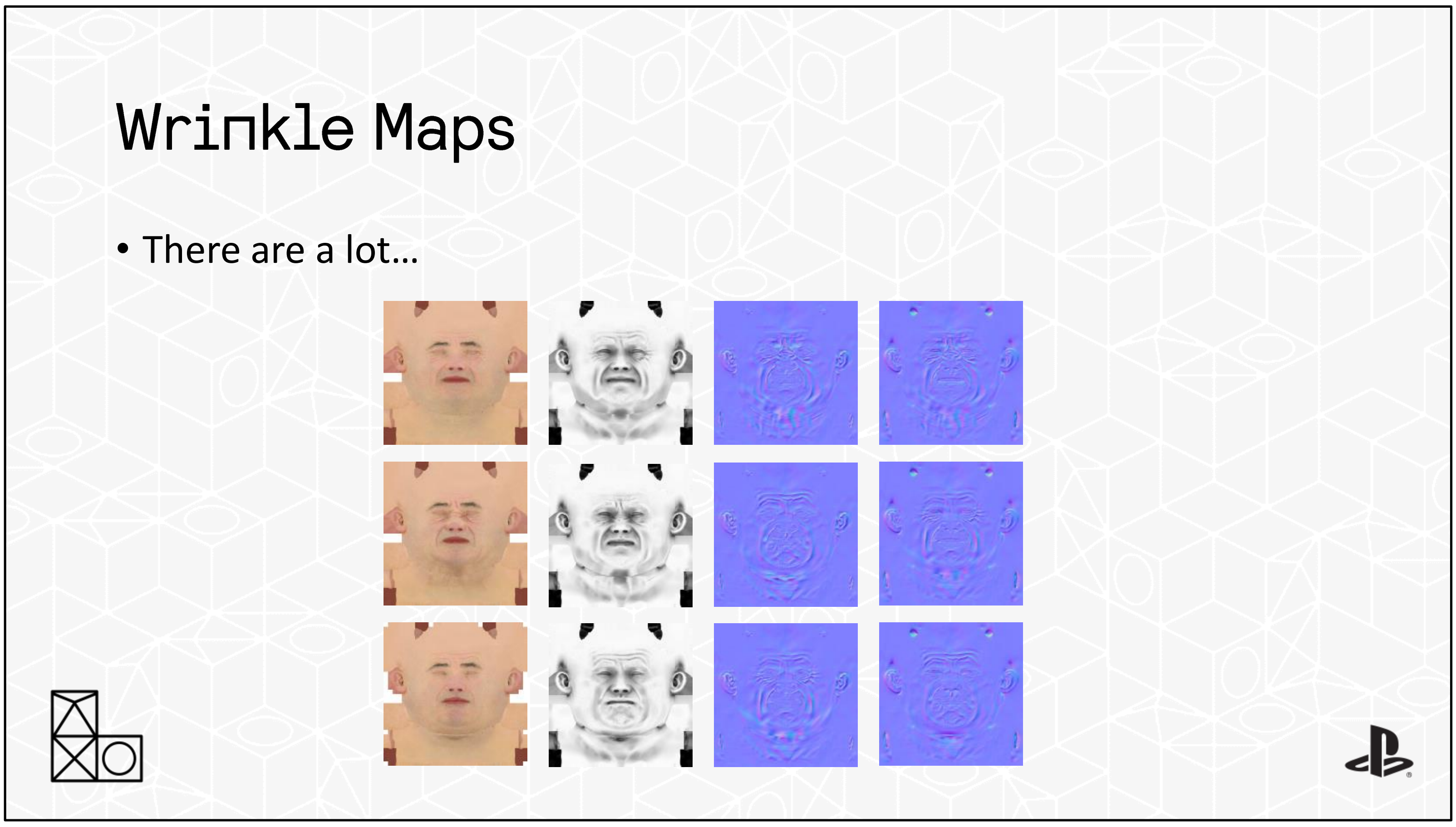
Bent Normals

- Micro Shadow [Brinck & Maximov 2016]



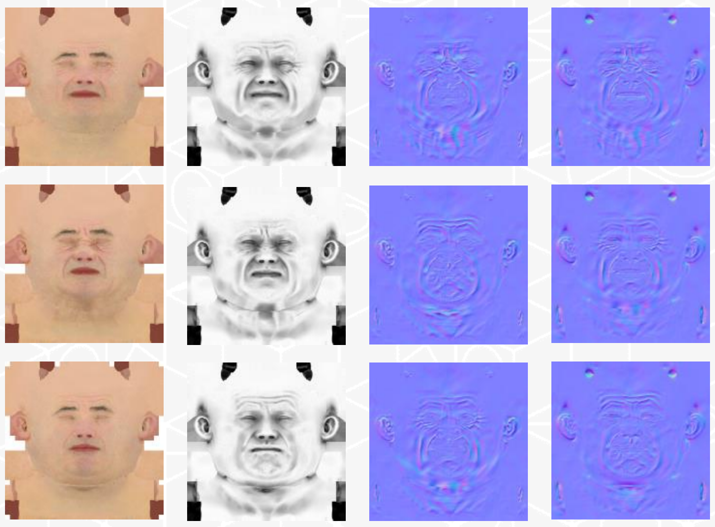


We define a cone of visibility with a direction from the bent normal and an aperture from the ambient occlusion value. Comparing the incoming light direction to this cone, we can get an estimate of how likely it is that this pixel is in shadow.
To show the effect of this more clearly, the direct light in this scene doesn't have shadows turned on. But in games we are unlikely to have perfect shadows anyway. You may have lights in the game with no shadows, low resolution shadow maps, or bias issues that leave holes. It's generally a good idea to make sure your character shading is robust enough to cope with less than ideal lighting scenarios.
It also allows us to self shadow details than exist purely in the textures rather than geometry, which is why we also store bent normals and occlusion for our wrinkle maps – it allows us to have some idea of self shadowing on the wrinkles as they come in.



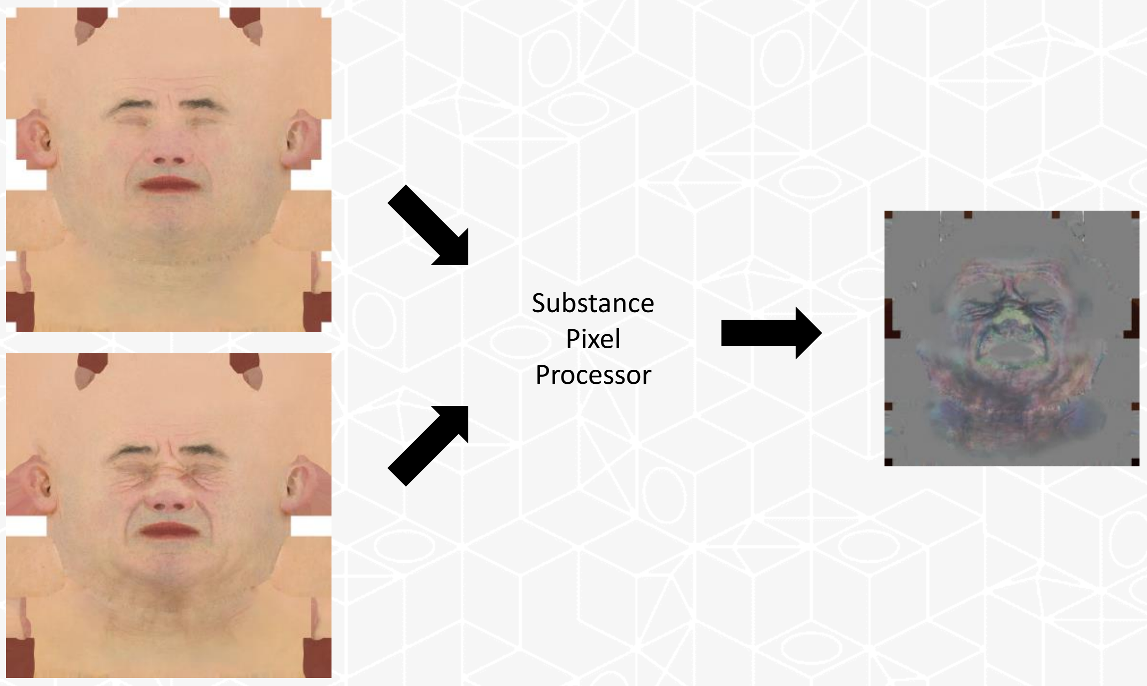
Wrinkle Maps

- There are a lot…



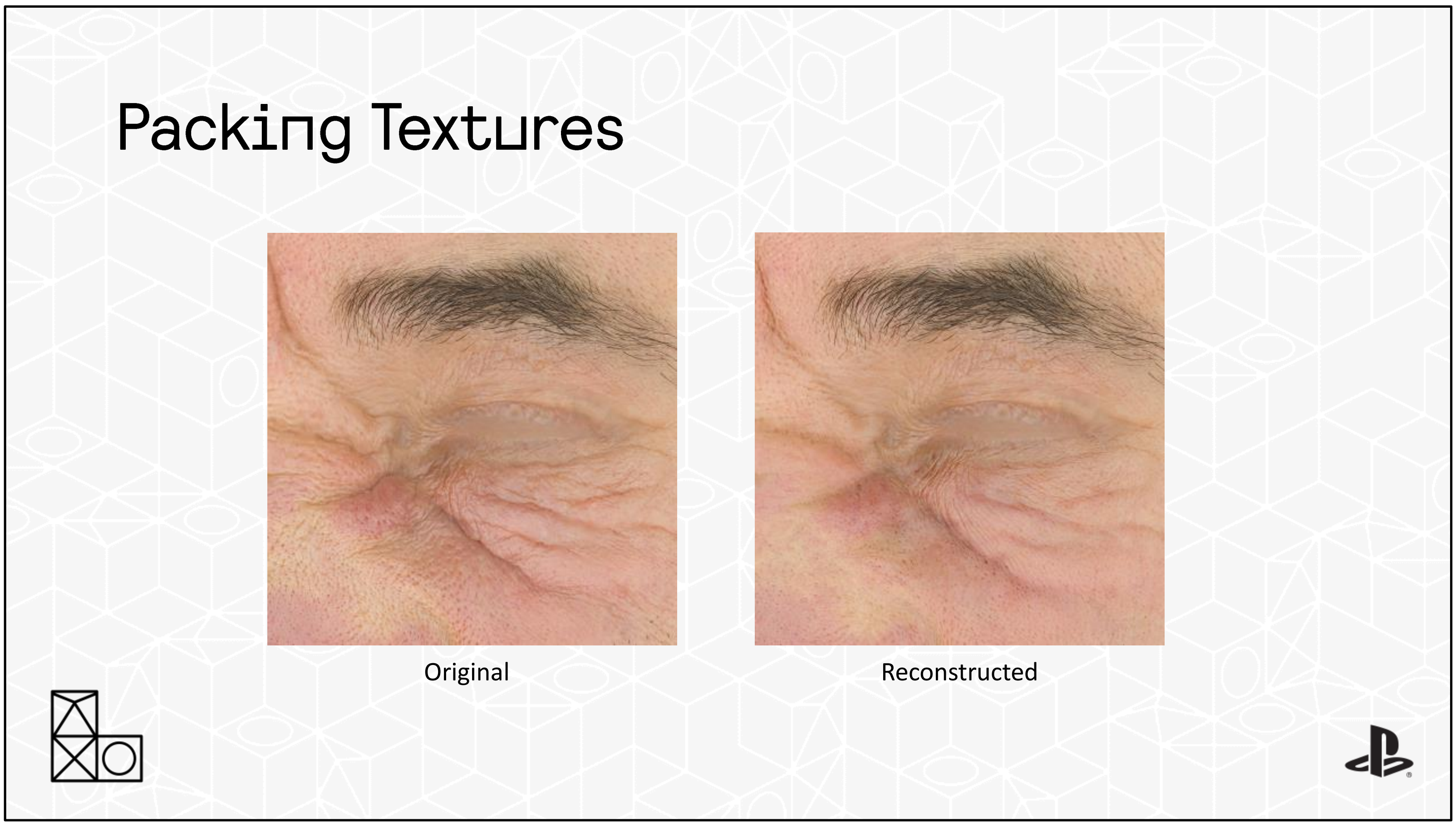
So as I've mentioned for wrinkle maps we change most of the properties, albedo, occlusion, normal, and bent normal. This does mean we have another 12 textures. Obviously this could take up a huge amount of extra memory if left unpacked, plus there will be an additional performance hit on reading them all.
Occlusion is the most straightforward to pack as it's a single channel at an already low resolution, we can simply pack the base occlusion and the 3 wrinkle maps into one 4 channel BC7 texture.

Packing Textures

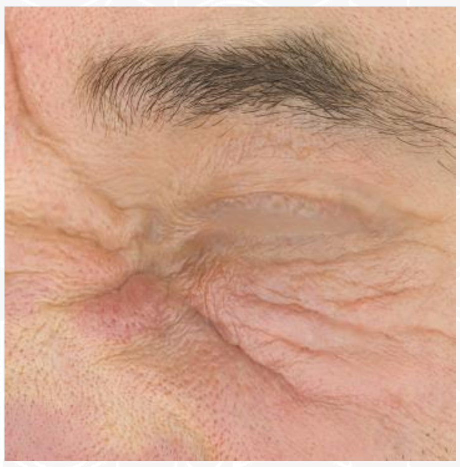


For the other maps, albedo, normal and bent normal, we decided that it was acceptable to only have coarse changes on wrinkles rather than fine detail. The wrinkle maps were processed to store the value differences from the base map, and downsampled. We stored the square root of the difference to maximize the colour range. To reconstruct we can just unpack and add to the base texture.

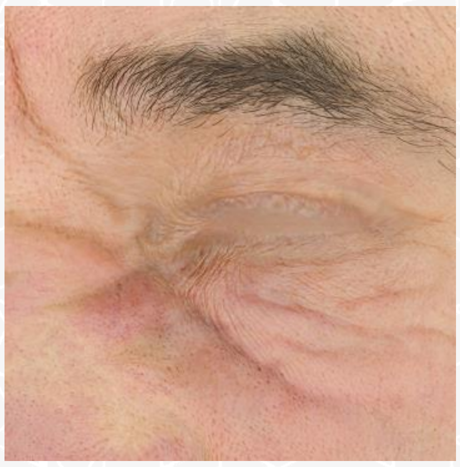
```
Storage pseudocode:
Difference = wrinkleTexture-baseTexture (both in linear space)
PackedResult = sqrt(abs(difference)) * sign(difference) * 0.5 + 0.5

Usage pseudocode:
Difference = differenceTexture * 2.0 - 1.0
Result = baseTexture + (difference * abs(difference) * wrinkleweight)
```



Packing Textures



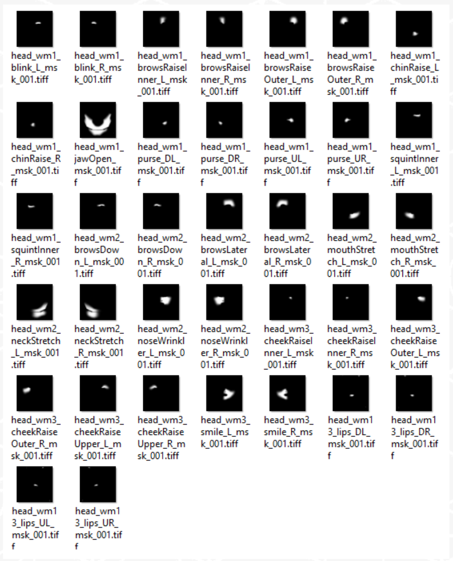
Original



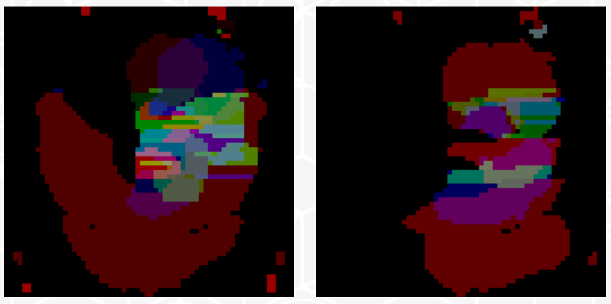
Reconstructed

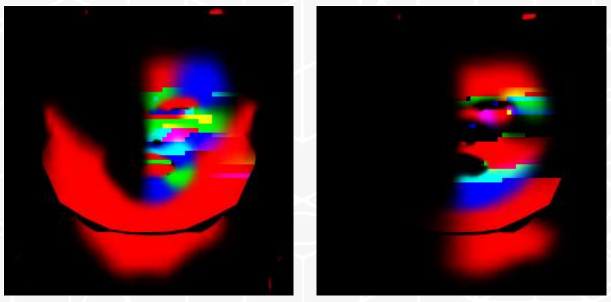
This shows the difference between the original wrinkle map and the reconstructed one using the base texture and the offsets. There is definitely a detail loss in some cases but we found it acceptable. We downsample the albedo to 512x512, stored as BC1 linear.
For the normals we actually reduce further, down to 256x256 stored as BC5. The reason for this is that we want to apply the offsets on top of the both the regular normals and the blurred ones we use for diffuse lighting, and we don’t want to reintroduce sharp features into the blurred normal.

Packing Textures

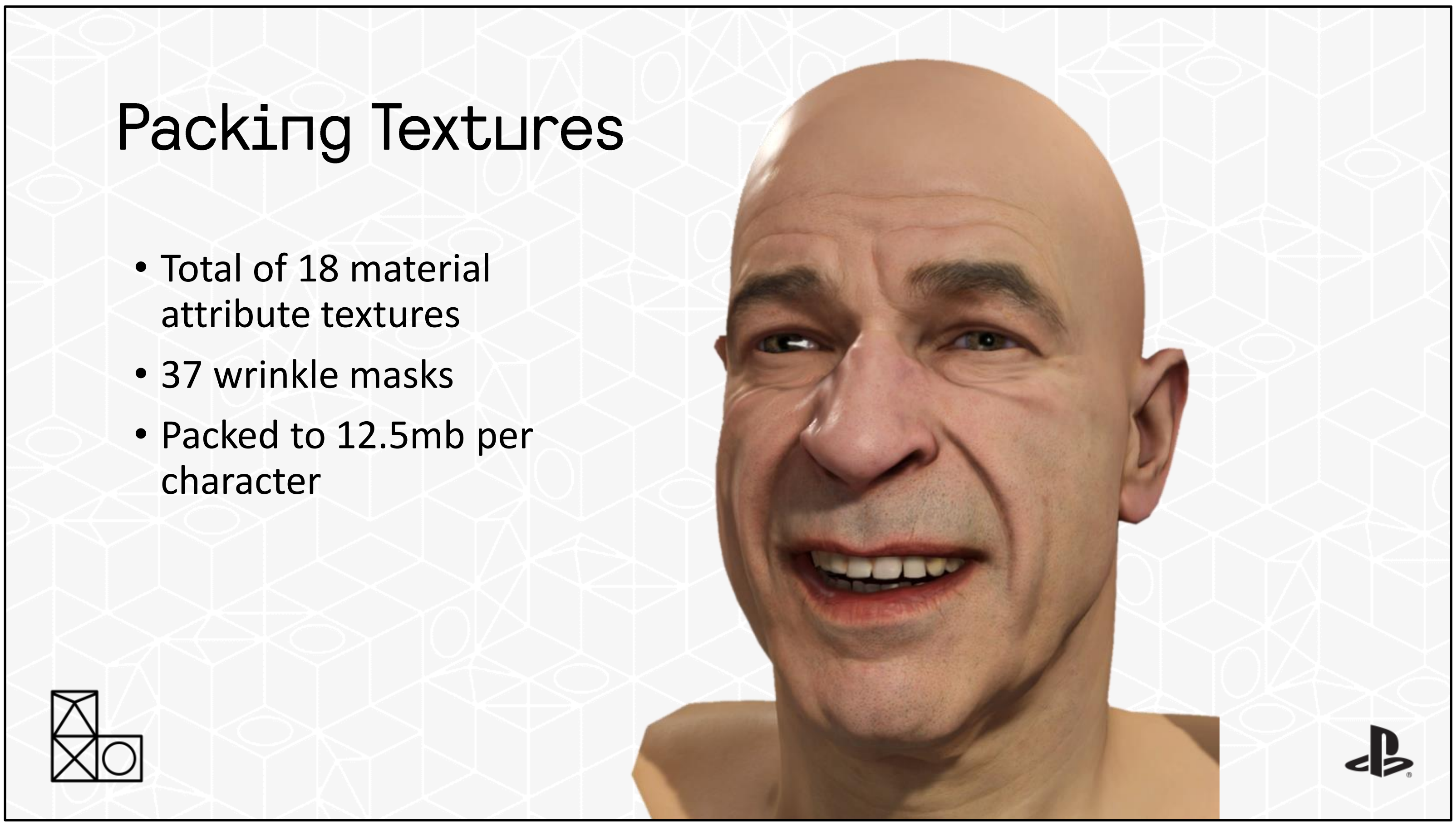


head_wm1_blink_L_msk_001.tiff
head_wm1_blink_R_msk_001.tiff
head_wm1_browsRaiseInner_L_msk_001.tiff
head_wm1_browsRaiseInner_R_msk_001.tiff
head_wm1_browsRaiseOuter_L_msk_001.tiff
head_wm1_browsRaiseOuter_R_msk_001.tiff
head_wm1_chinRaise_L_msk_001.tiff
head_wm1_chinRaise_R_msk_001.tiff
head_wm1_jawOpen_msk_001.tiff
head_wm1_purse_DL_msk_001.tiff
head_wm1_purse_DR_msk_001.tiff
head_wm1_purse_UL_msk_001.tiff
head_wm1_purse_UR_msk_001.tiff
head_wm1_squintInner_L_msk_001.tiff
head_wm1_squintInner_R_msk_001.tiff
head_wm2_browsDown_L_msk_001.tiff
head_wm2_browsDown_R_msk_001.tiff
head_wm2_browsLateral_L_msk_001.tiff
head_wm2_browsLateral_R_msk_001.tiff
head_wm2_mouthStretch_L_msk_001.tiff
head_wm2_mouthStretch_R_msk_001.tiff
head_wm2_neckStretch_L_msk_001.tiff
head_wm2_neckStretch_R_msk_001.tiff
head_wm2_noseWrinkler_L_msk_001.tiff
head_wm2_noseWrinkler_R_msk_001.tiff
head_wm3_cheekRaiseInner_L_msk_001.tiff
head_wm3_cheekRaiseInner_R_msk_001.tiff
head_wm3_cheekRaiseOuter_L_msk_001.tiff
head_wm3_cheekRaiseOuter_R_msk_001.tiff
head_wm3_cheekRaiseUpper_L_msk_001.tiff
head_wm3_cheekRaiseUpper_R_msk_001.tiff
head_wm3_smile_L_msk_001.tiff
head_wm3_smile_R_msk_001.tiff
head_wm13_lips_DL_msk_001.tiff
head_wm13_lips_DR_msk_001.tiff
head_wm13_lips_UL_msk_001.tiff
head_wm13_lips_UR_msk_001.tiff


Index textures - 64x64


Coverage textures – 256x256

From the face rig we had 37 mask channels. We packed them with a system similar to skinning weights, using 2 64x64 RGBA textures to store indices of the masks, with up to 8 overlapping. We then had coverage values for each of these, stored at 256x256.

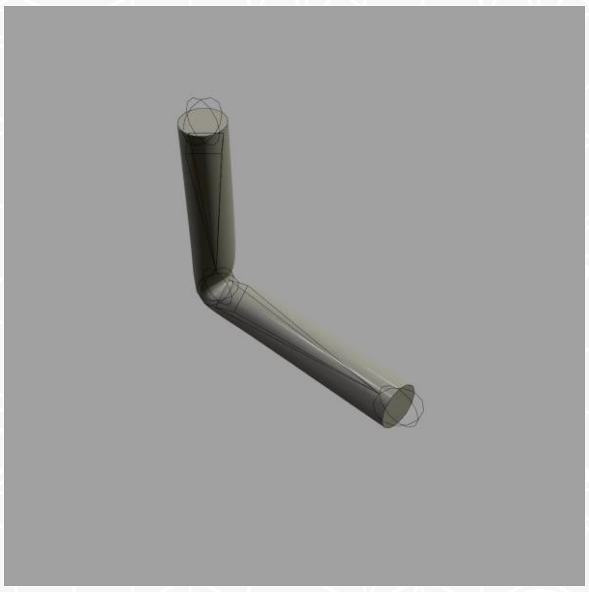


Packing Textures

- Total of 18 material attribute textures
- 37 wrinkle masks
- Packed to 12.5mb per character

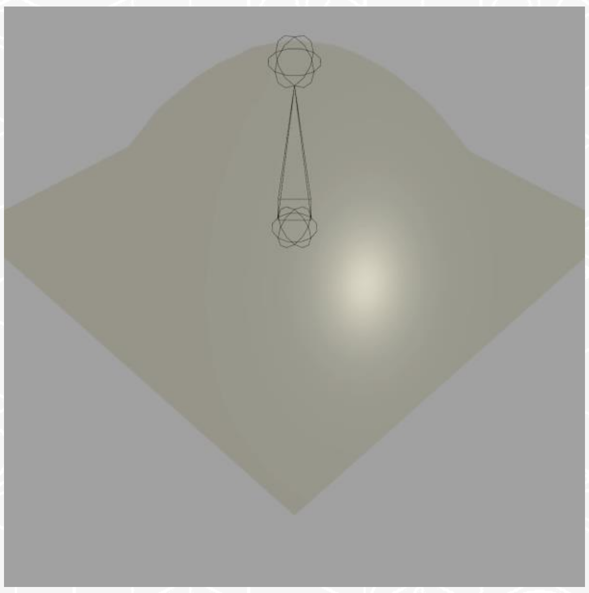
In the end we pack 18 material textures and 37 masks into 12.5mb of memory, whilst maintaining all the wrinkle data we wanted.

Normal Recalculation

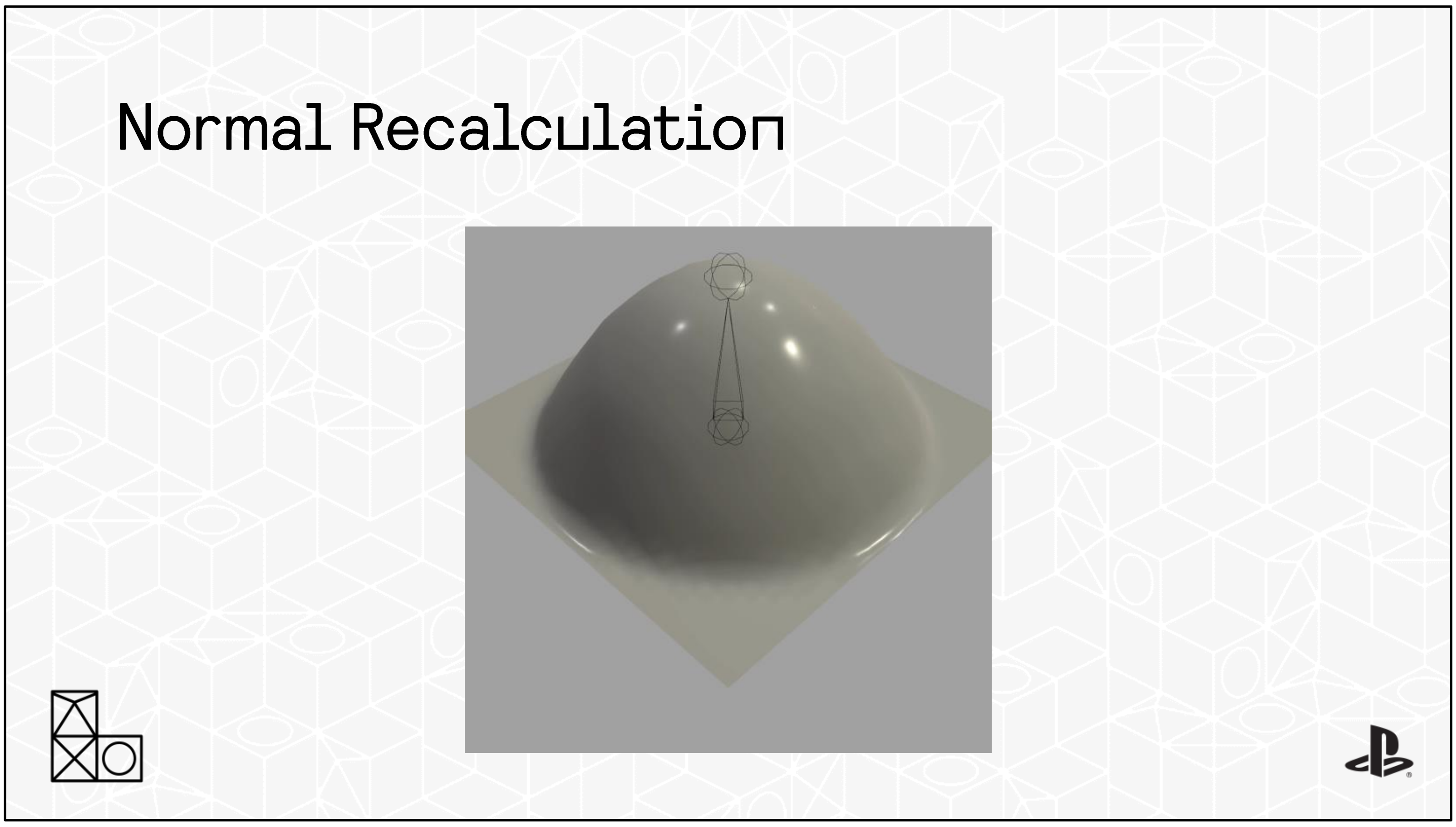


Next I want to talk about normal recalculation. We used an entirely joint based rig on Blood & Truth, and it was important to recalculate the normals on the fly to make sure the lighting looked correct. Standard game skinning only rotates the normals with the bone. This works fine for a lot of common movement, like limbs. As you can see here, the normals of this cylinder are rotating along with the bone and it looks fine.

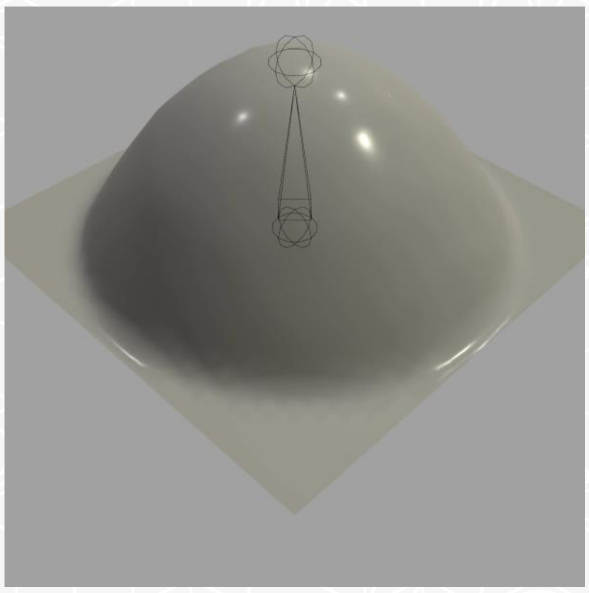
Normal Recalculation



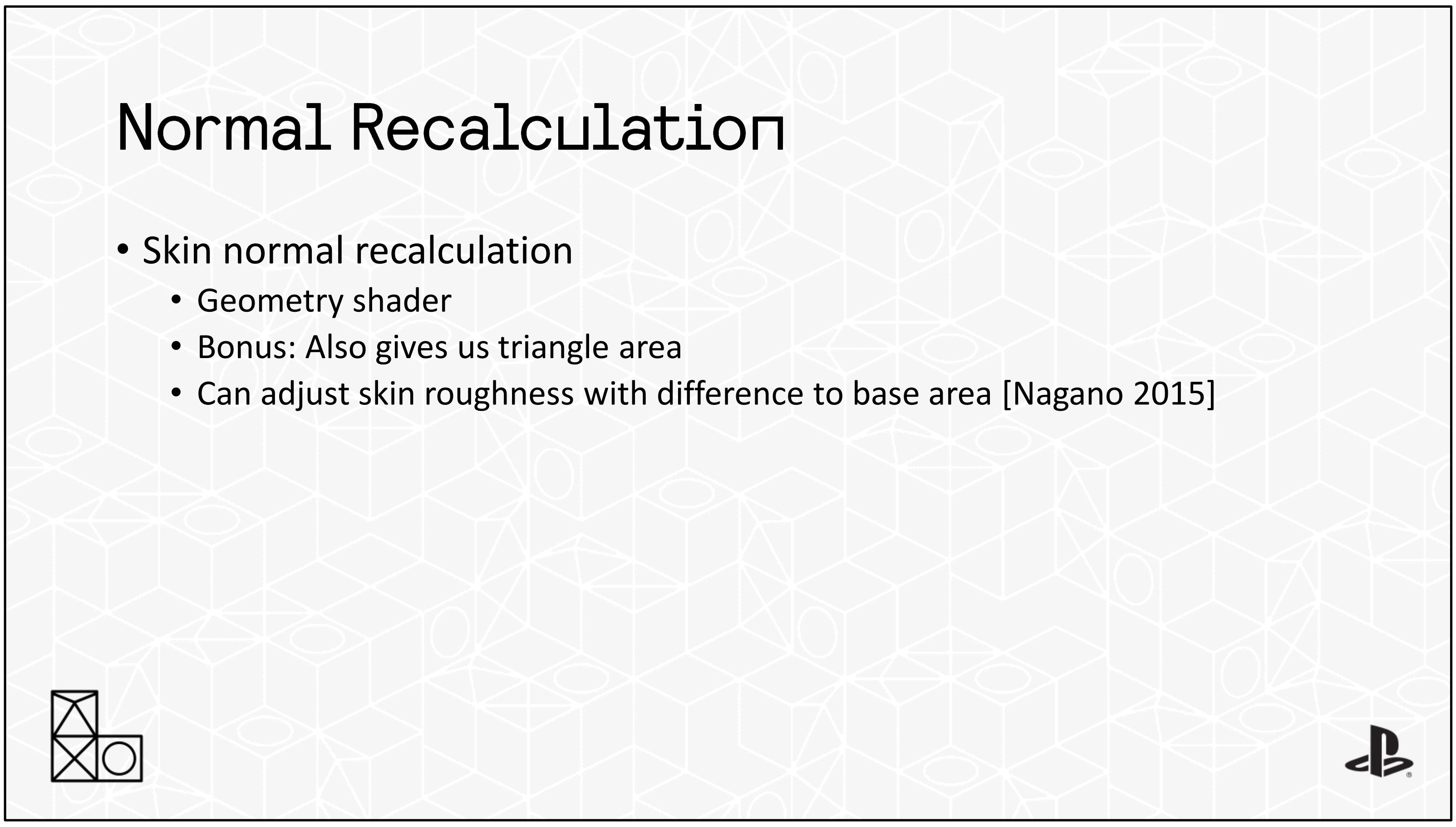
Unfortunately, this doesn’t take into account any new shapes created by joint translation. In this example, we should be seeing a bulge, but because the normals are entirely flat there is no change in shading.



Normal Recalculation



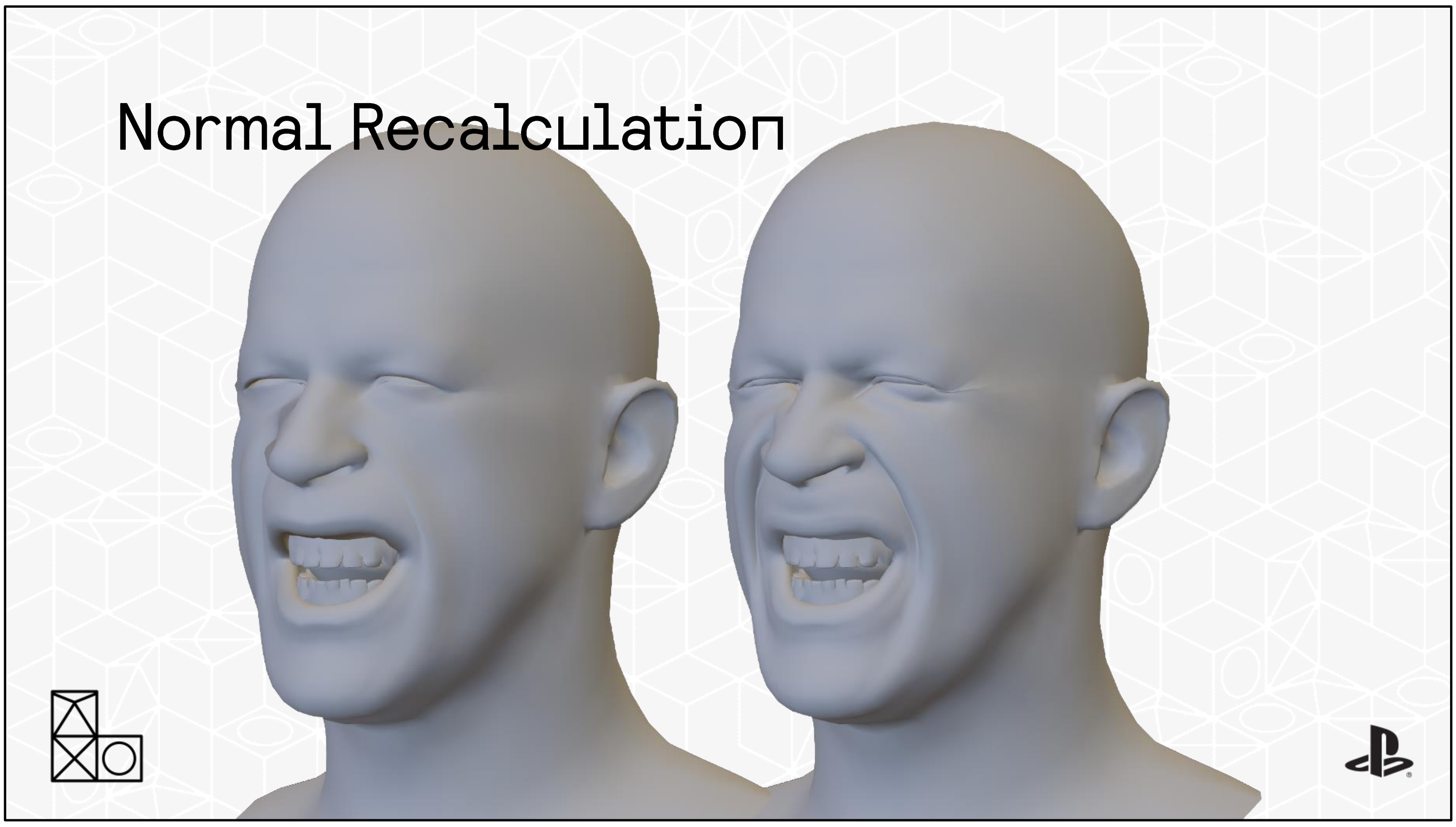
If we generate new normals by calculating the normals on the surrounding faces and averaging them it looks correct. This is really important for facial animation, as most of the joints are translating rather than rotating as they move on the face. It makes a large difference to being able to read the facial expression of the characters as intended.



Normal Recalculation

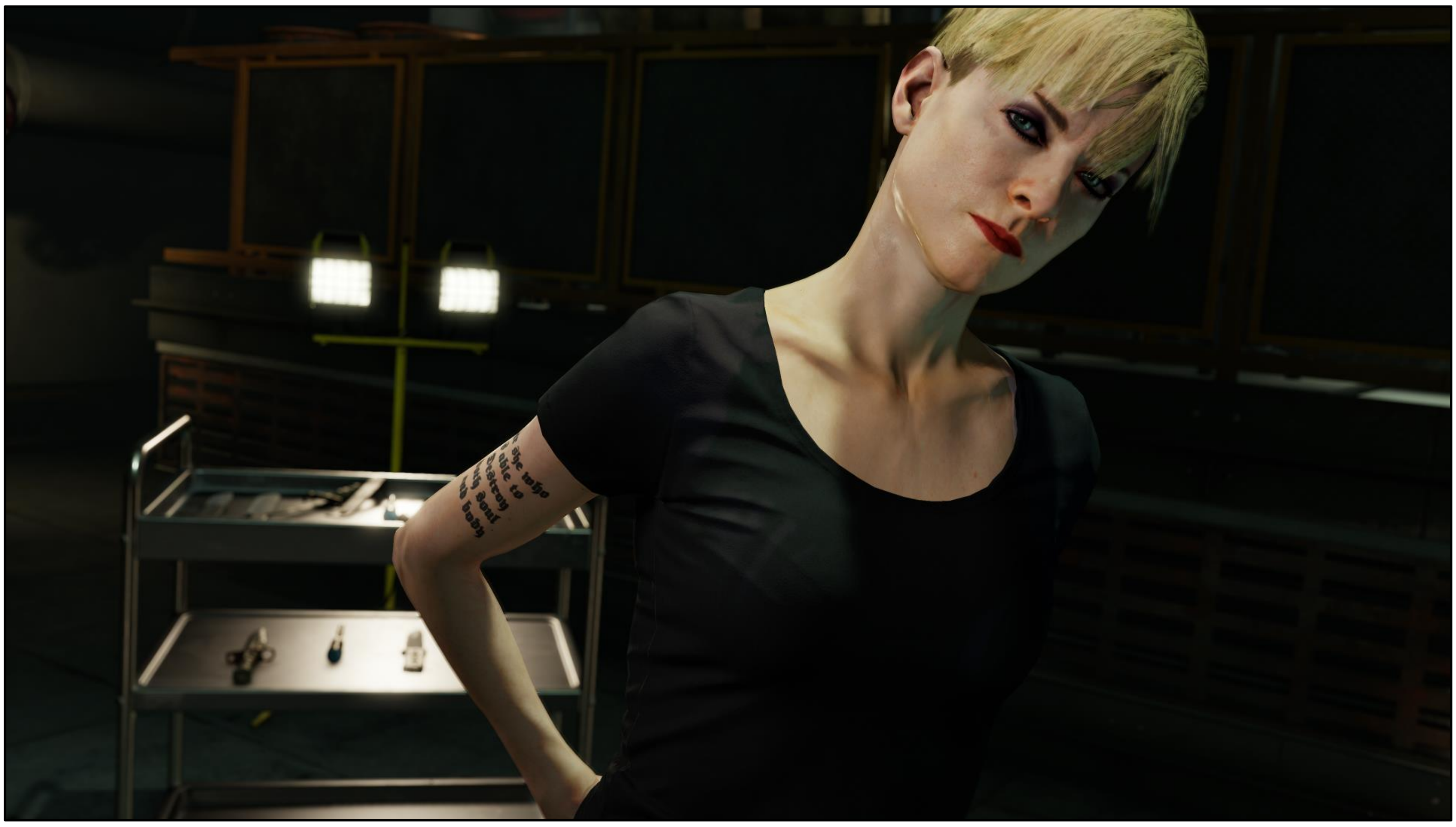
- Skin normal recalculation
  - Geometry shader
  - Bonus: Also gives us triangle area
  - Can adjust skin roughness with difference to base area [Nagano 2015]

For Blood & Truth we used a geometry shader to do this process on the faces. One other benefit is that it’s very easy to calculate the area of the triangles on the face in the same process. By comparing this area against a stored value from the base pose, we can get some idea of the squash and stretch of the face. We were inspired by Koki Nagano’s work studying the microstructure changes of skin as it deforms. As an art-driven ad-hoc approximation to the effect, we simply adjusted the roughness of the skin based on our squash and stretch value.

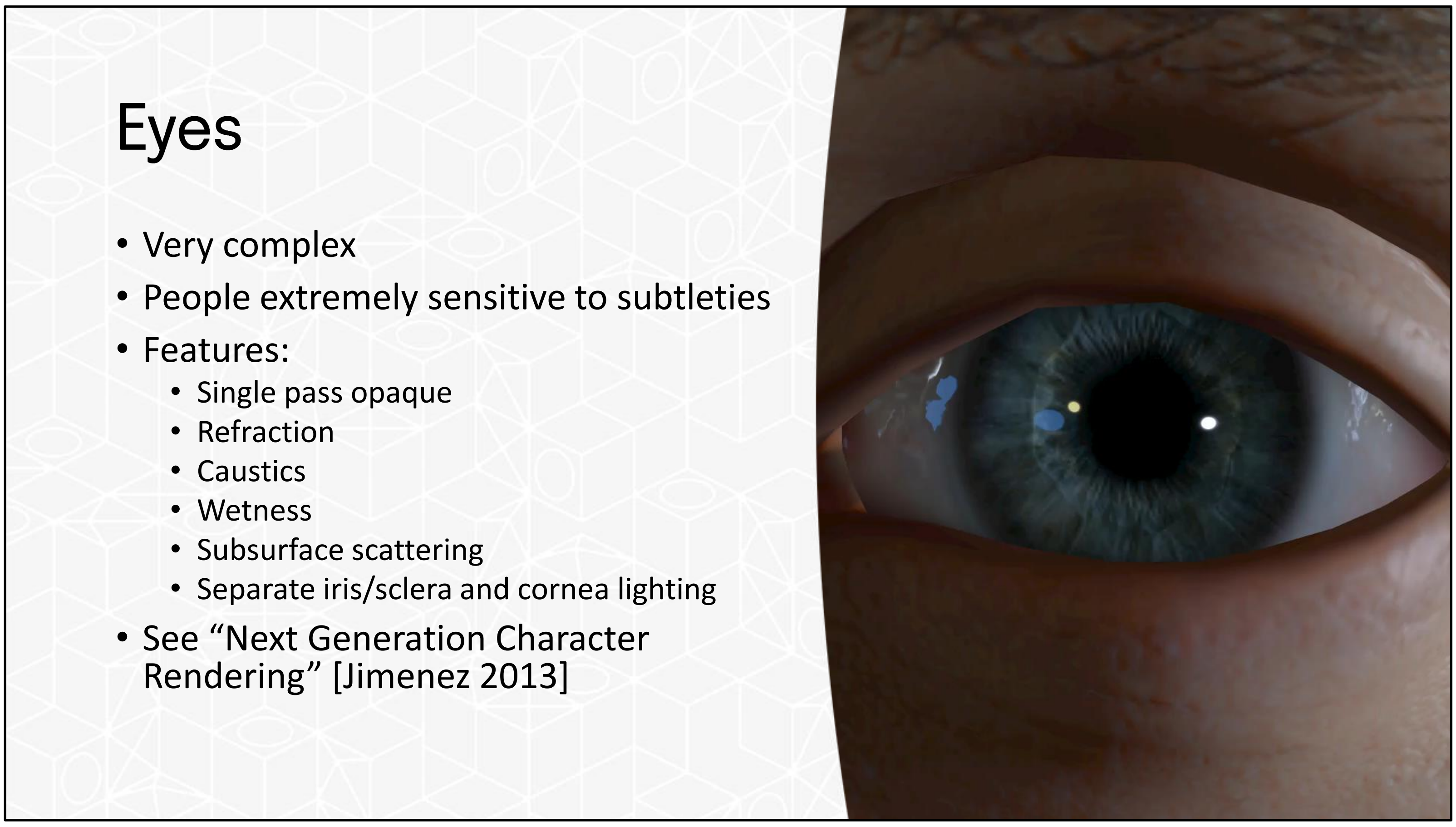


Normal Recalculation

You can see the difference that normal recalculation makes in this comparison. It’s especially apparent on lip pursing and wrinkles that are formed by geometry changes.

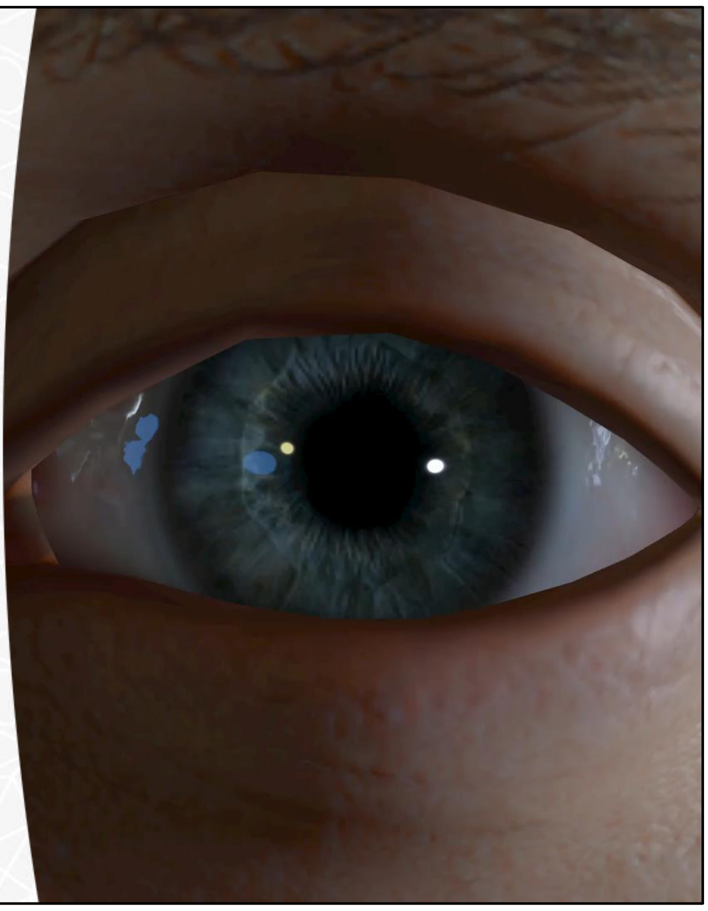


The result of all of this is that we have a high quality skin shader that all runs in a single pass using our forward renderer.

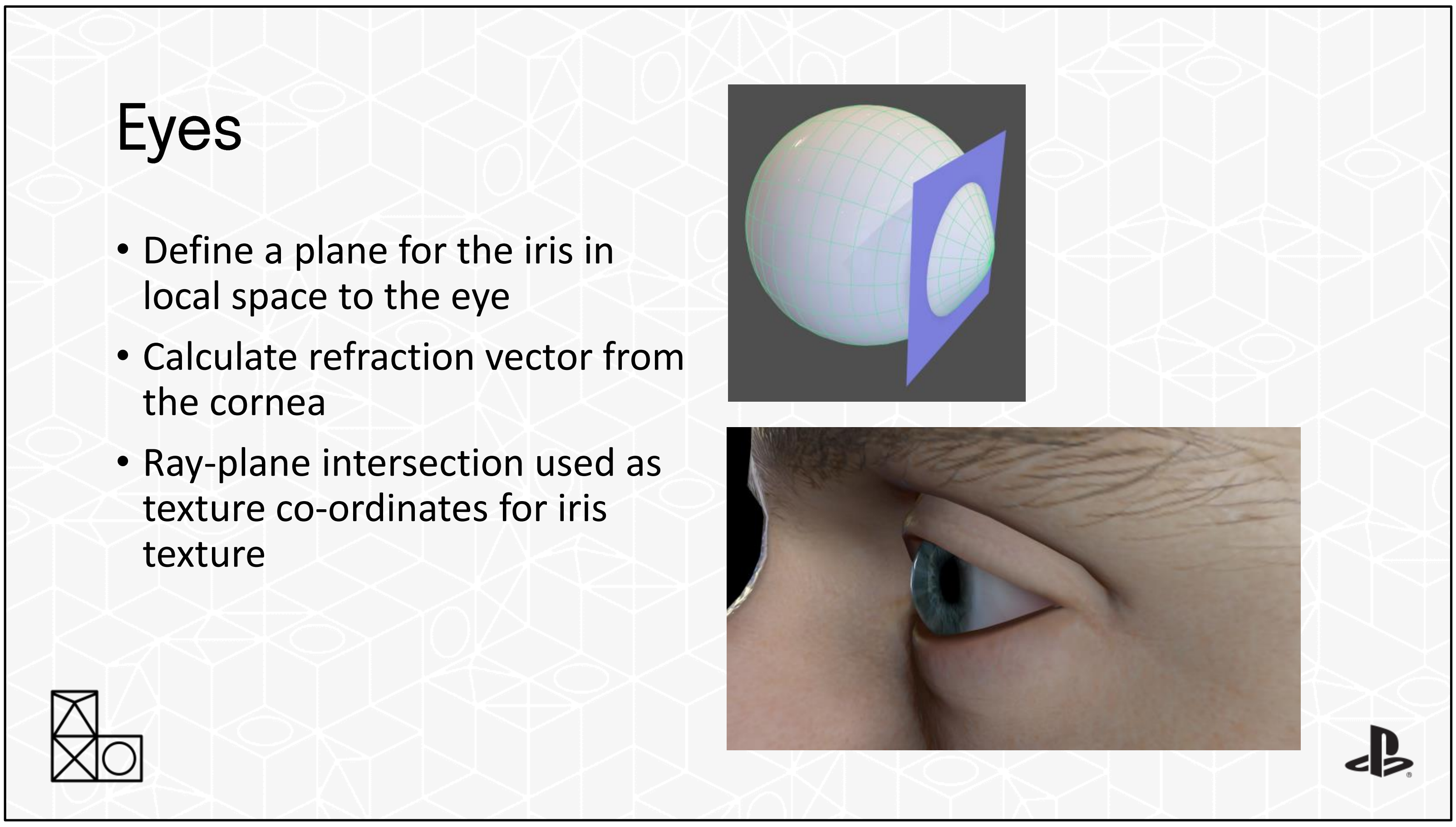


Eyes

- Very complex
- People extremely sensitive to subtleties
- Features:
  - Single pass opaque
  - Refraction
  - Caustics
  - Wetness
  - Subsurface scattering
  - Separate iris/sclera and cornea lighting
- See “Next Generation Character Rendering” [Jimenez 2013]

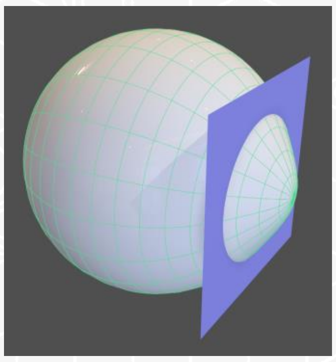


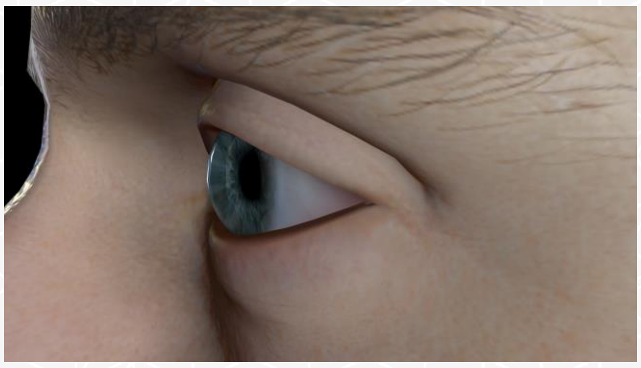
The next subject I’m going to cover today is the eyes. Human eyes are very complex multi layered materials that we are also extremely used to seeing. As with the skin, we wanted to make a material which could run in a single pass without transparencies. We simulate a number of effects in this pass including refraction, caustics and subsurface scattering. For a good breakdown of the component parts of eye rendering, I’d highly recommend Jimenez’s talk “Next Generation Character Rendering”



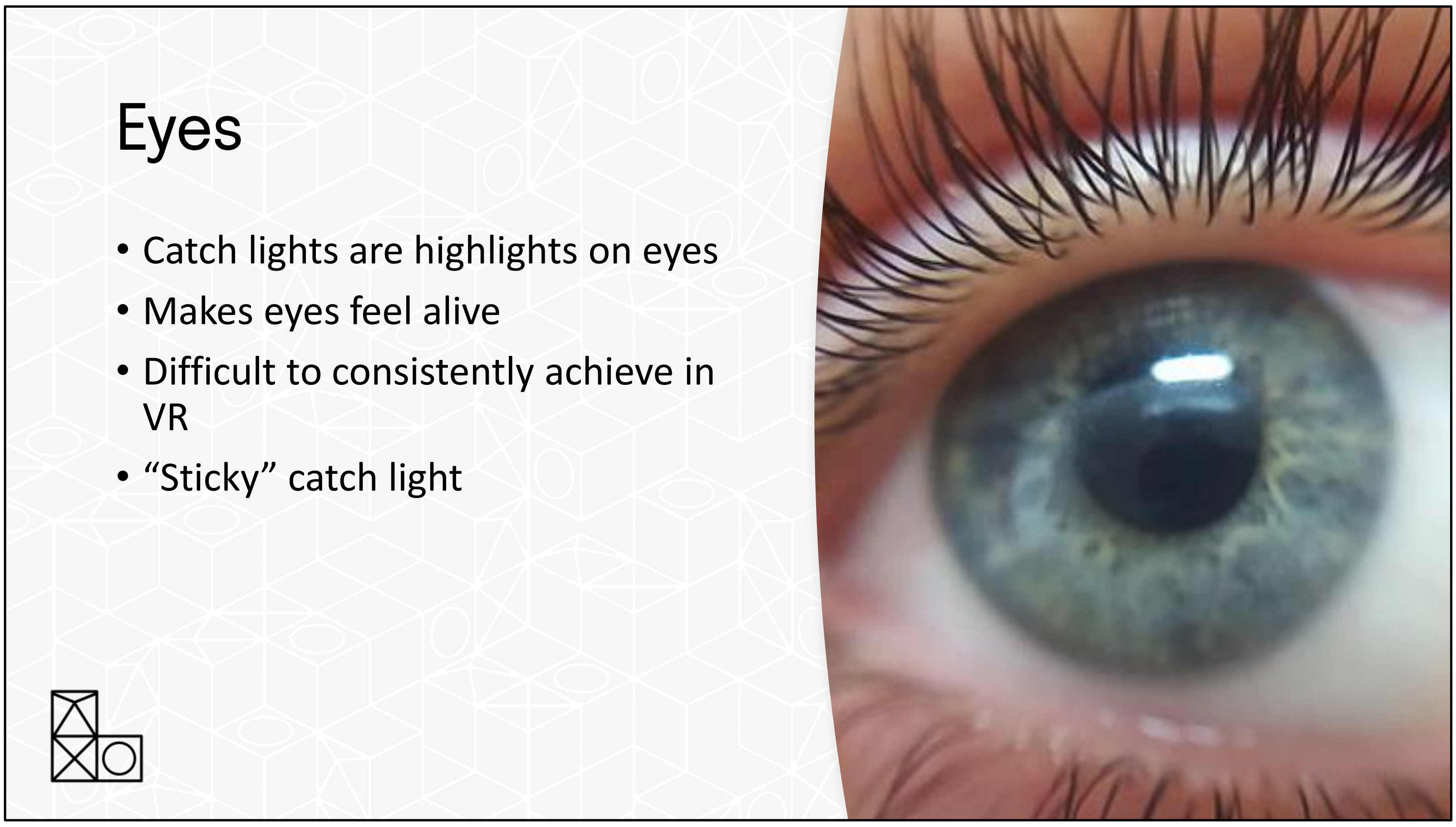
Eyes

- Define a plane for the iris in local space to the eye
- Calculate refraction vector from the cornea
- Ray-plane intersection used as texture co-ordinates for iris texture





As an example of what we can do in this single pass, rather than having the cornea as a refractive layer of geometry on top of the iris, we can calculate the effect directly in the shader. We take the refraction vector from the cornea and intersect an imaginary plane representing the iris. We can then use the intersection point as a texture coordinate for the iris texture.

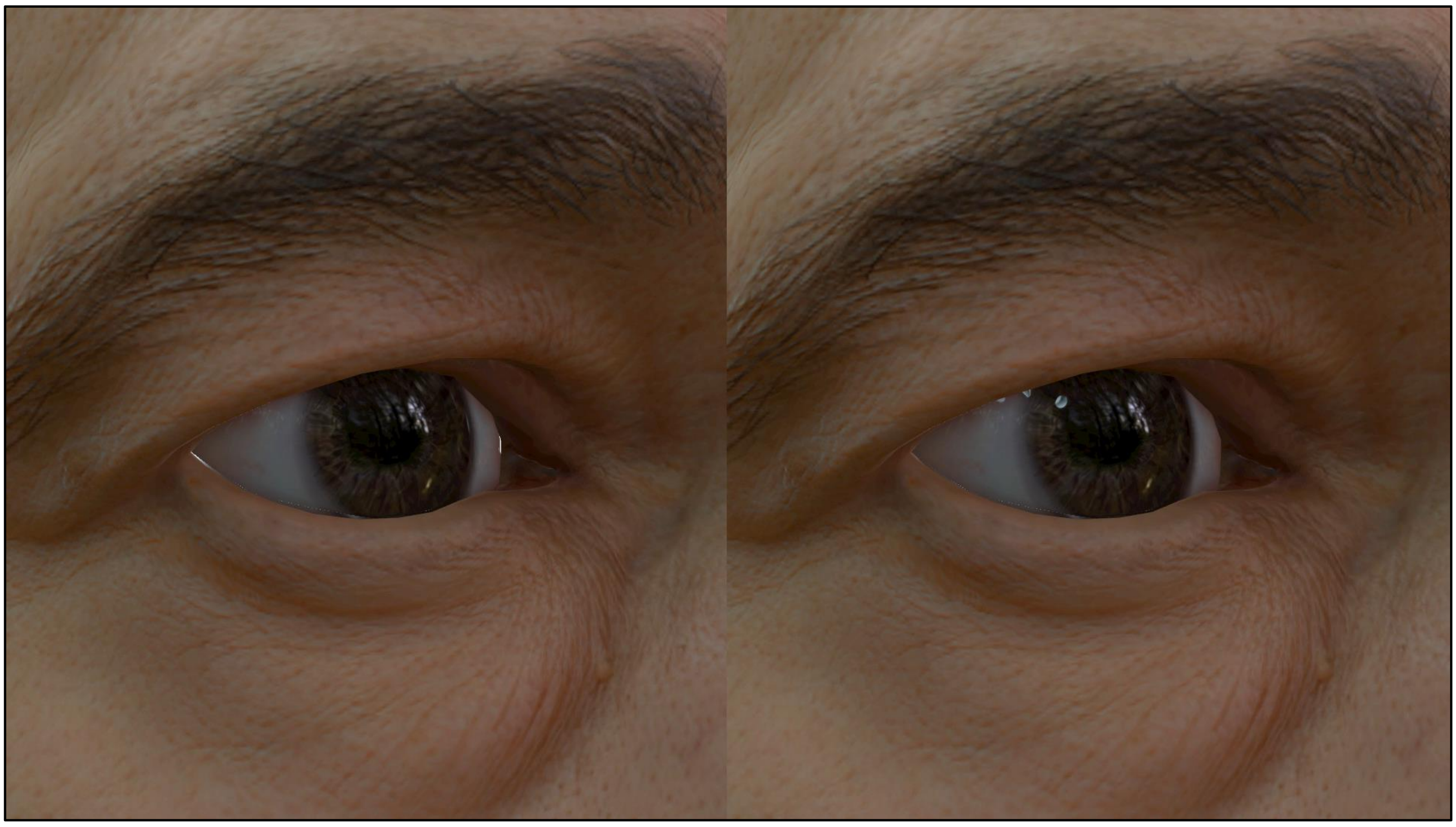


Eyes

- Catch lights are highlights on eyes
- Makes eyes feel alive
- Difficult to consistently achieve in VR
- “Sticky” catch light

In photography and film, lighting artists will usually try to get some kind of strong specular highlight on the eye, as it makes it feel feel much more alive. Game cutscenes often employ similar techniques, as we have a similar level of control over per shot lighting, and can move lights as needed.

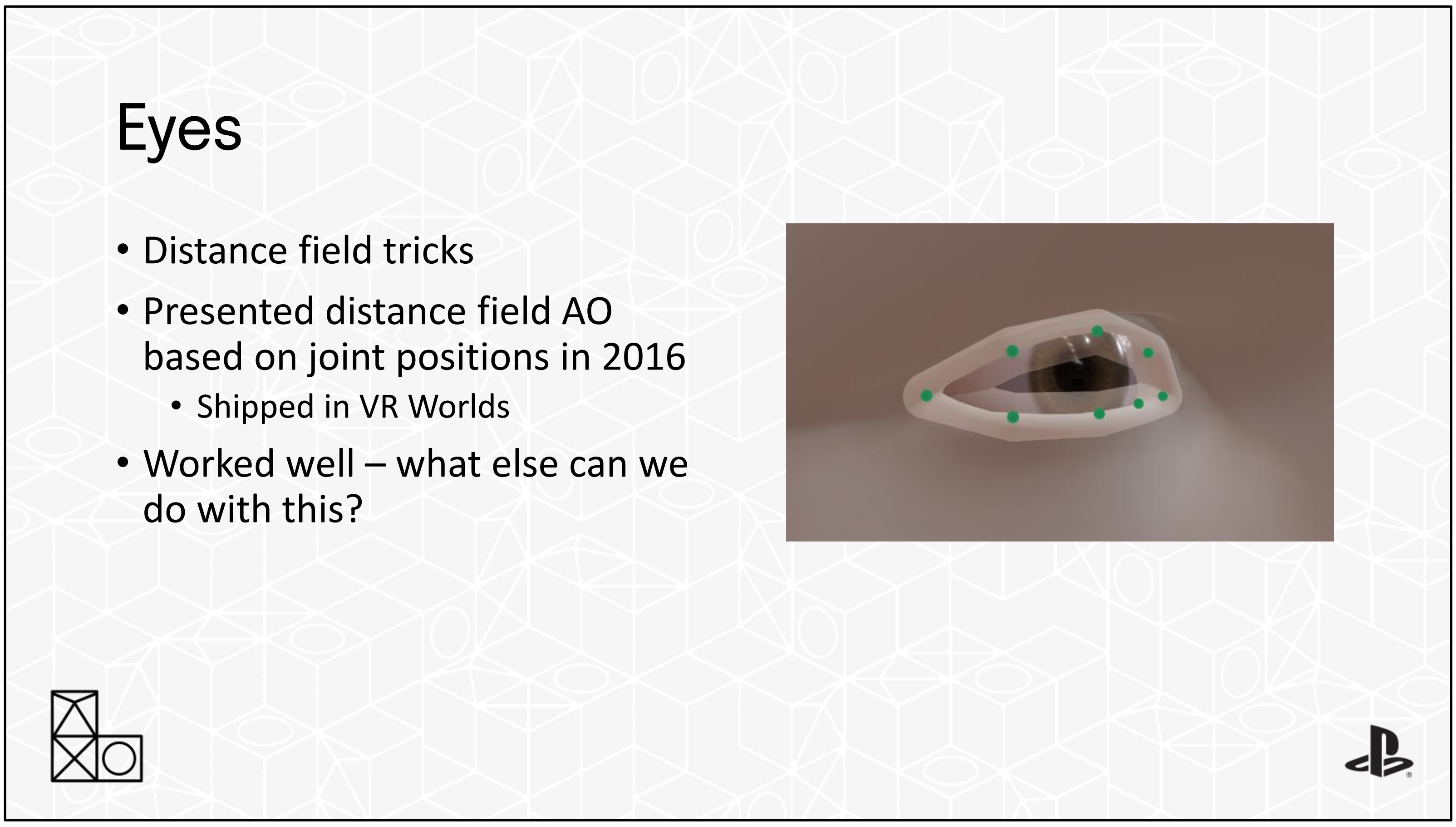
For gameplay, and for VR, we are much more limited. In Blood and Truth although we have some character specific lighting, without camera cuts to hide lighting changes we are more limited in changing the setup as the character moves around. As a solution for this, we have something which we call “sticky” catch lights, which is to say lights whose specular highlight sticks to the visible area of the eyeball.



On the left we have regular lighting, and on the right is sticky catch lights. You can see that although the specular highlight still moves with the light direction, it sticks in the visible area of the iris. This is accomplished by biasing the specular light vector towards the view vector, when it gets within a plausible range.
This means we get a more pleasing look on the eyes more consistently, and also means that we don't have to link any specific lights to the eyes. It also means that you have more freedom in creating the light and shadows you want on the face without worrying about maintaining the eye highlight.

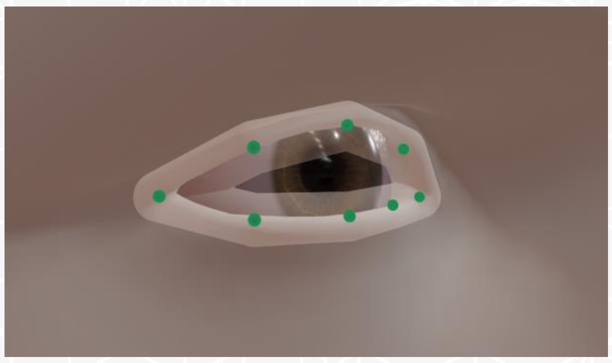
Code:

```
float hackFactor = saturate(-dot(L, -V)*-2.0-1.0);
L = normalize(lerp(L, V, saturate(0.4-hackFactor*hackFactor)));
```



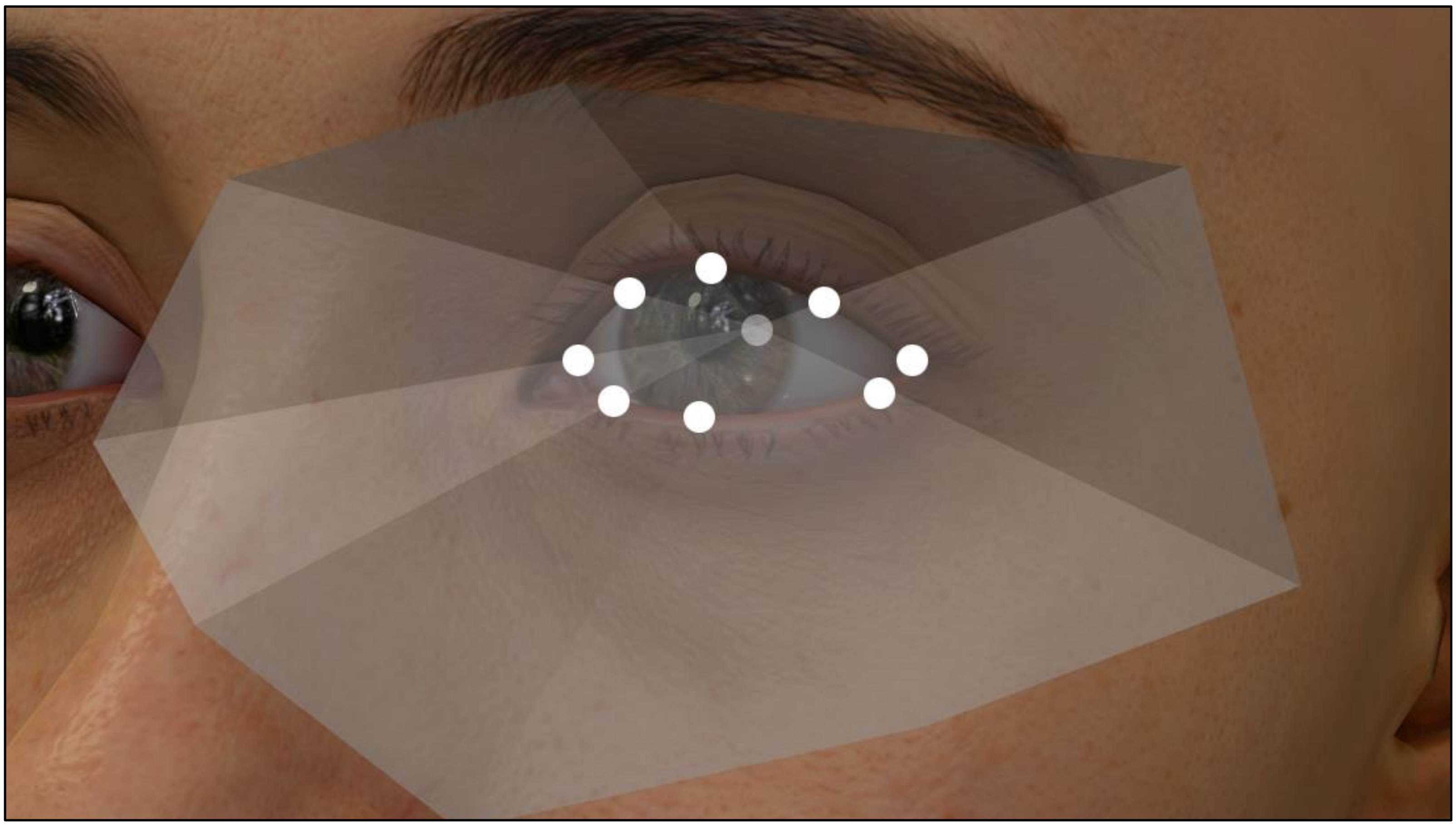
Eyes

- Distance field tricks
- Presented distance field AO based on joint positions in 2016
  - Shipped in VR Worlds
- Worked well – what else can we do with this?

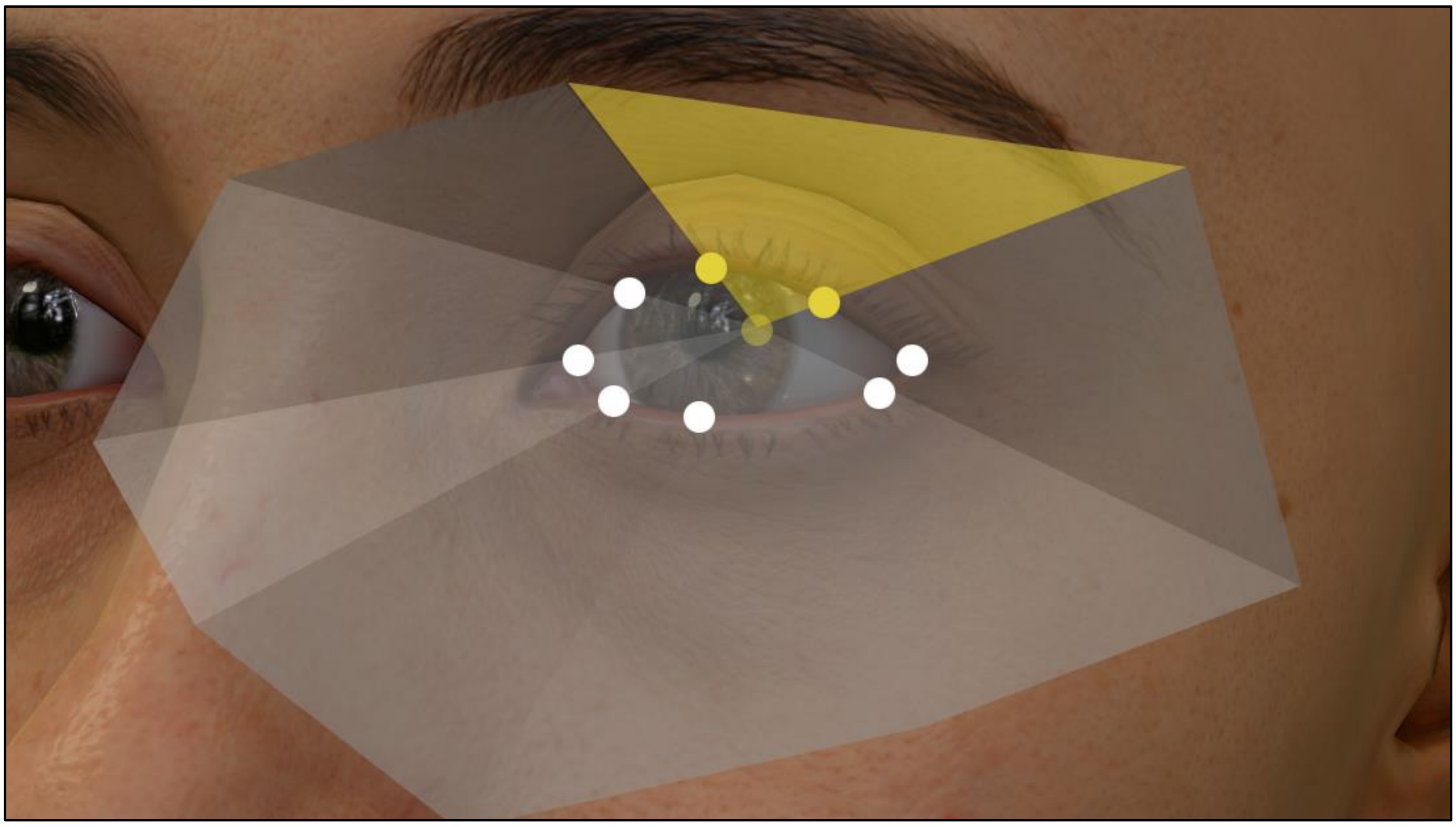


In VR Worlds we used a technique to generate ambient occlusion for the eyes by generating a distance field based on the joint positions of the eyelids, within the eye shader. This helped maintain shadowing on the eyes during times where screen space ambient occlusion might fail due to lack of resolution or scenes where it wasn't required.

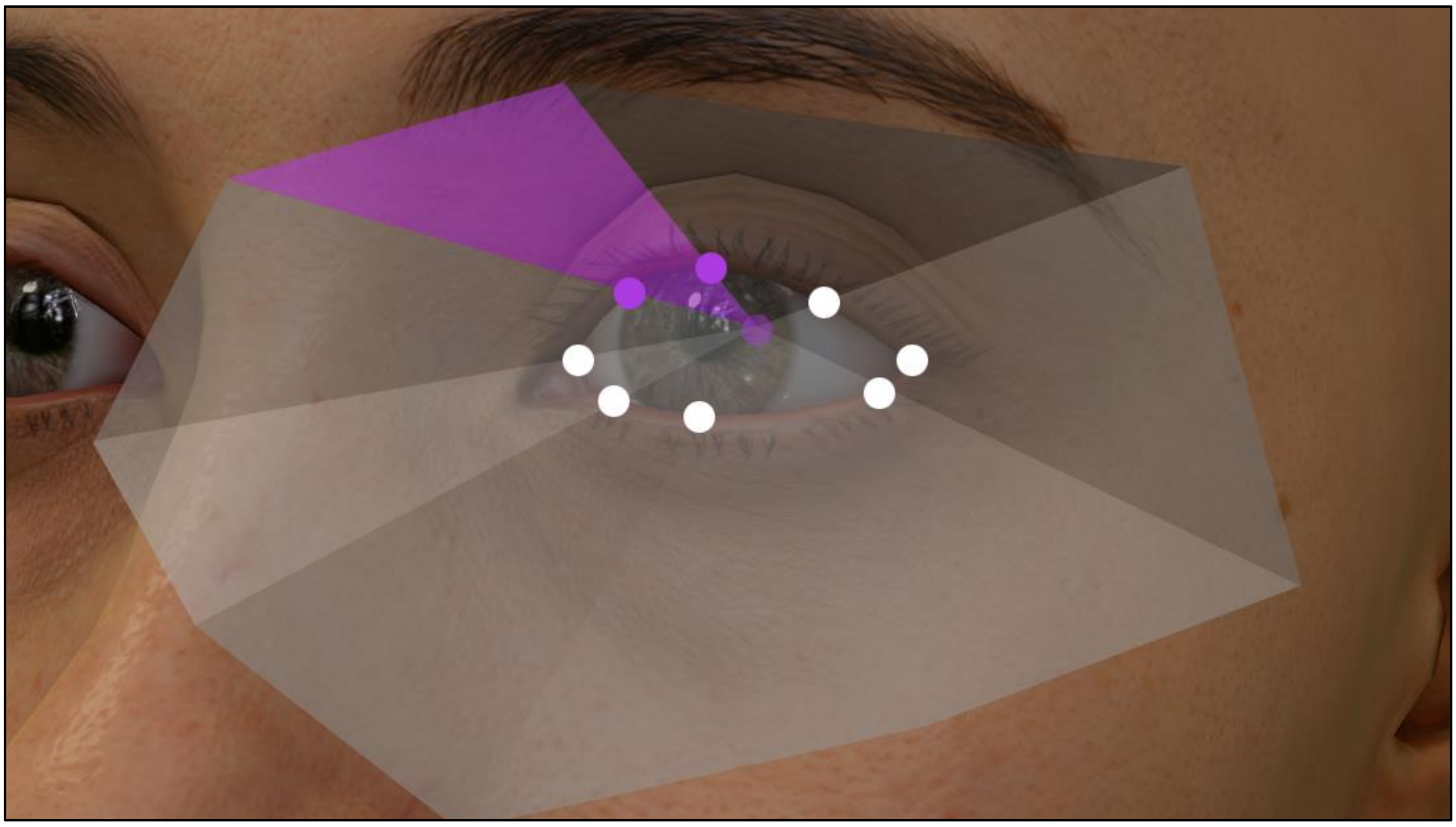
For Blood & Truth we extended this technique to simulate more eye rendering features on top of occlusion



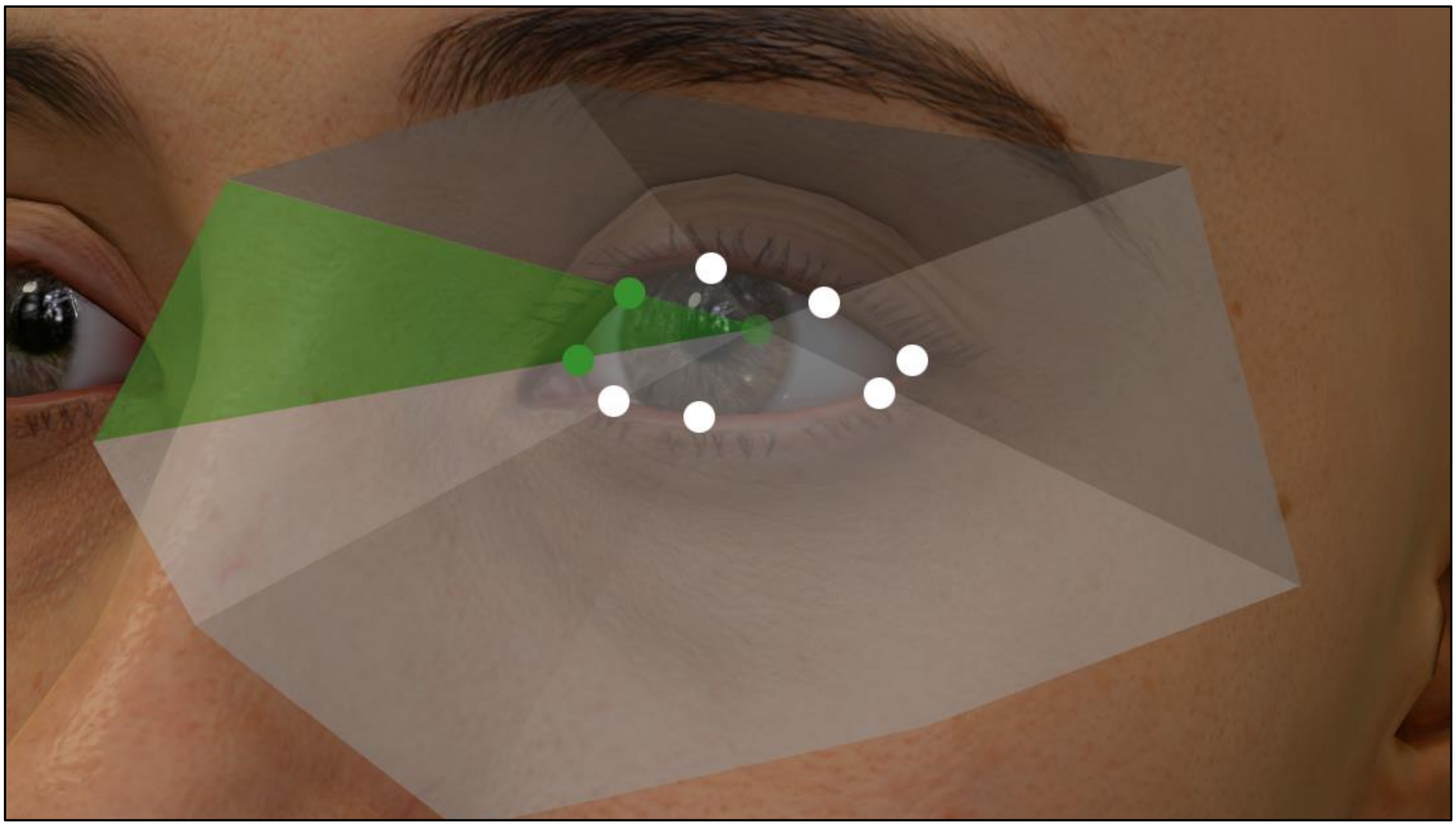
We take 8 eyelid joints on the character skeleton, which were already present on the skeleton for skinning the eyelids to. We pass these joint positions through to the eye shader. Within the fragment shader, we generate virtual planes defined by these joint positions, and calculate the distance to them.



So here we’ve defined a plane with these two eyelid joints and the centre of the eye

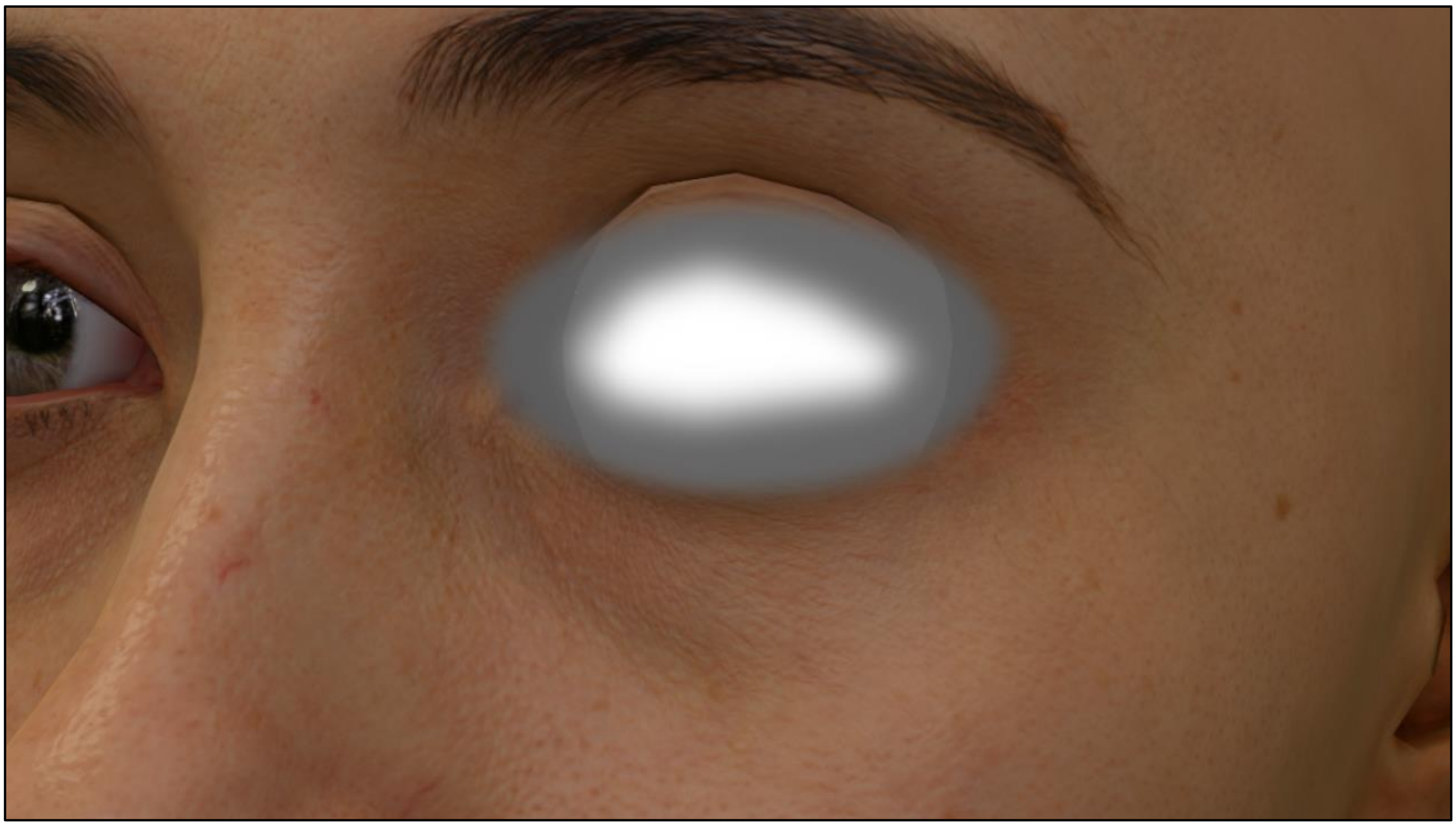


And the next one

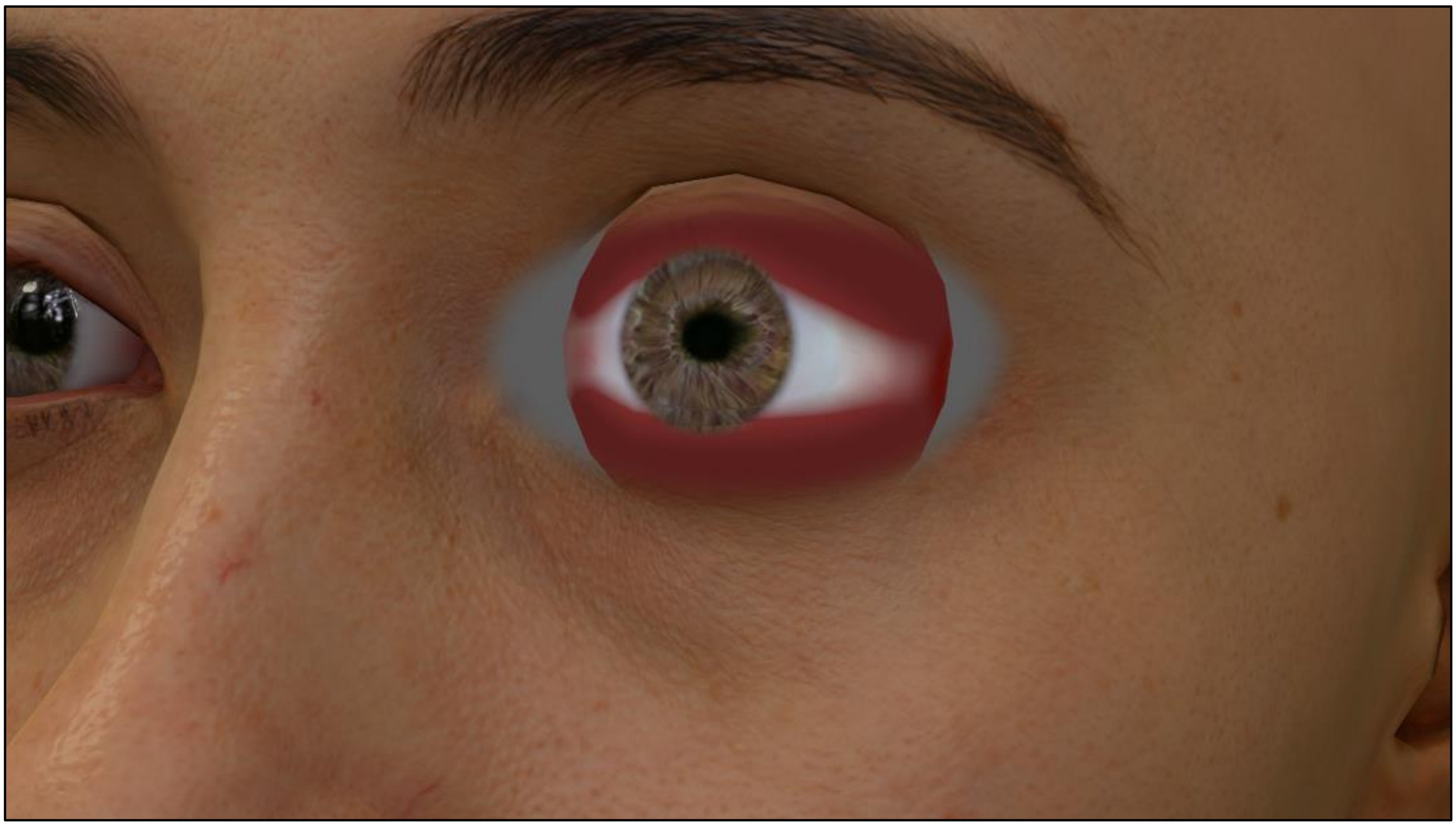


And so on, you get the idea.

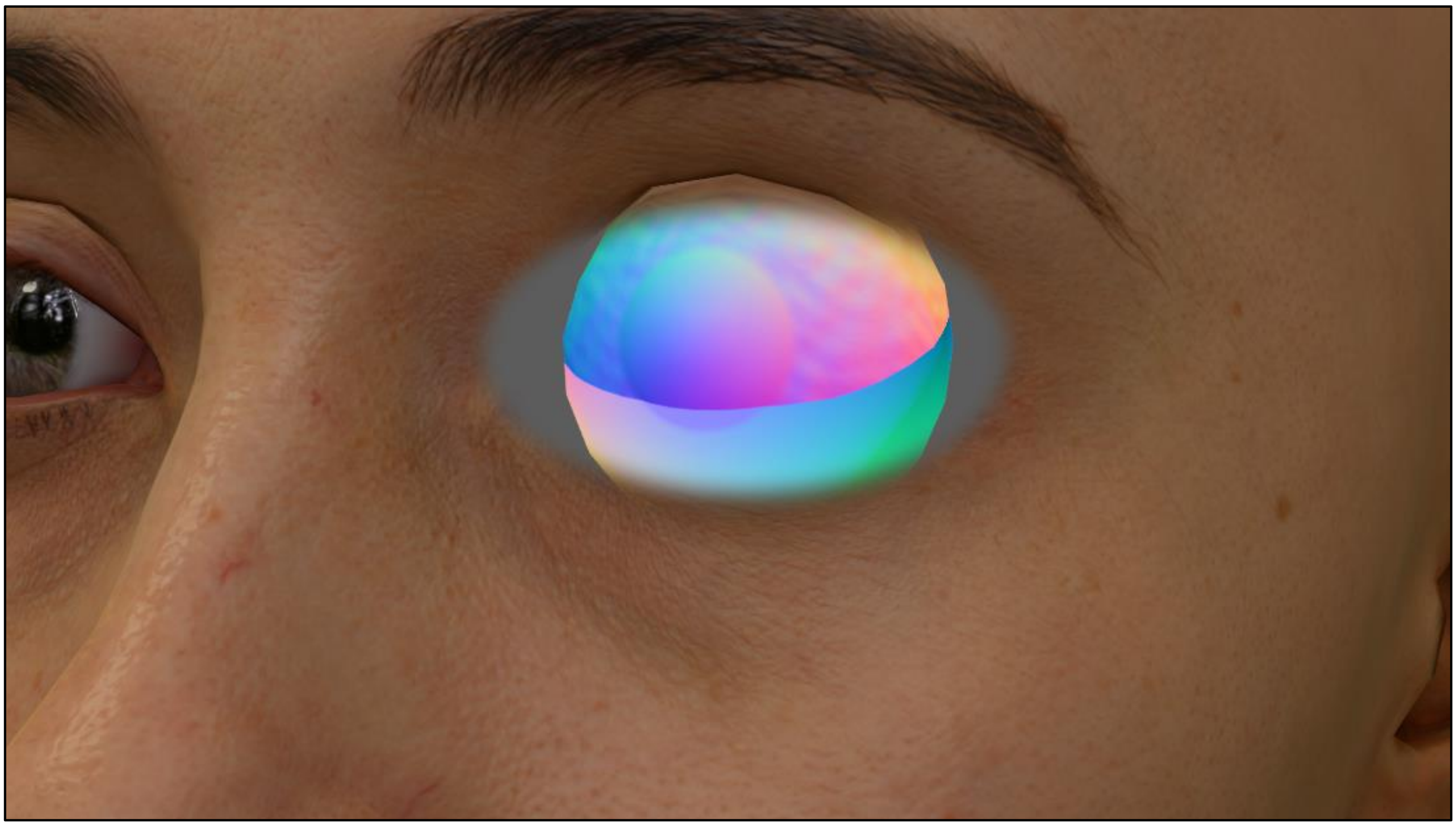
By calculating the distance to these 8 planes and blending we can get a sort of cone shape defined by the eyelids and the centre of the eye



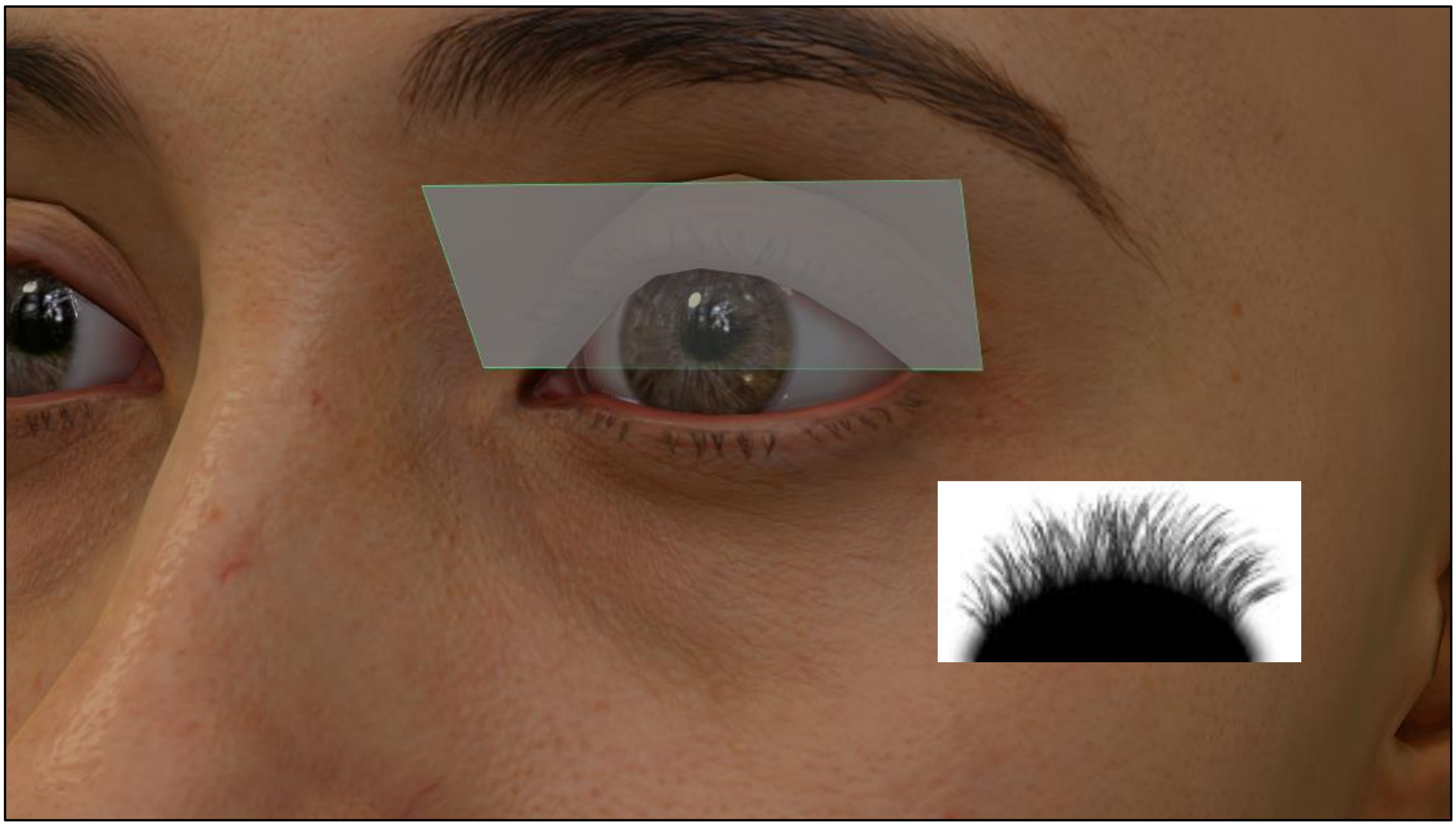
Remapping the distance to an occlusion value gets a fairly good approximation to AO. This is pretty close to what shipped in VR Worlds, but we wanted to see what other things we could do with the same technique.



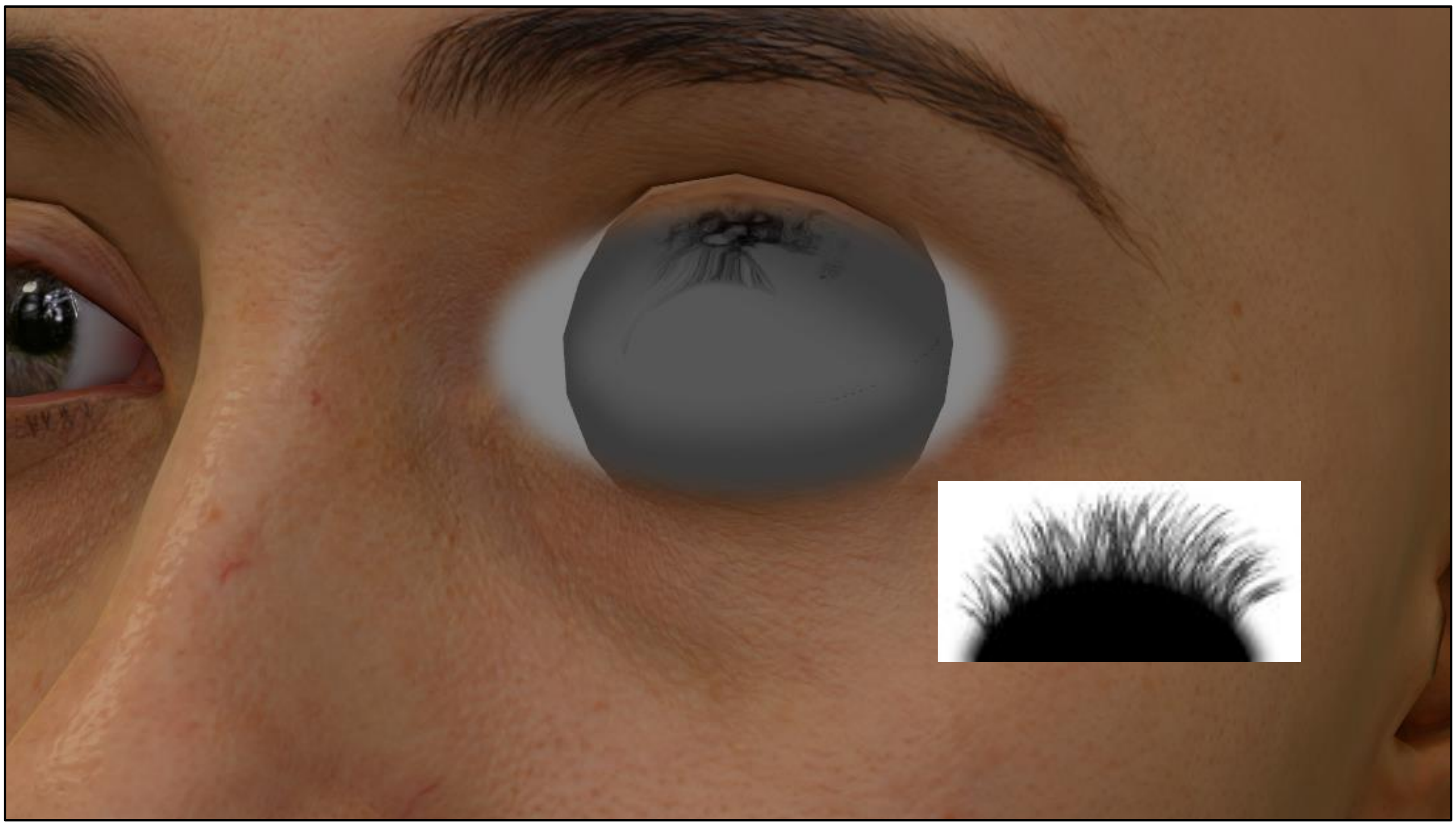
One nice effect some eye rendering systems have is blurring the transition from eyelid to skin using a post process blur. We can't do that in this case, but we can change the colour of the albedo to simulate some of the scattered and bounced light around the transition area. We also use the distance from the tear duct joint to create a caruncle transition effect.



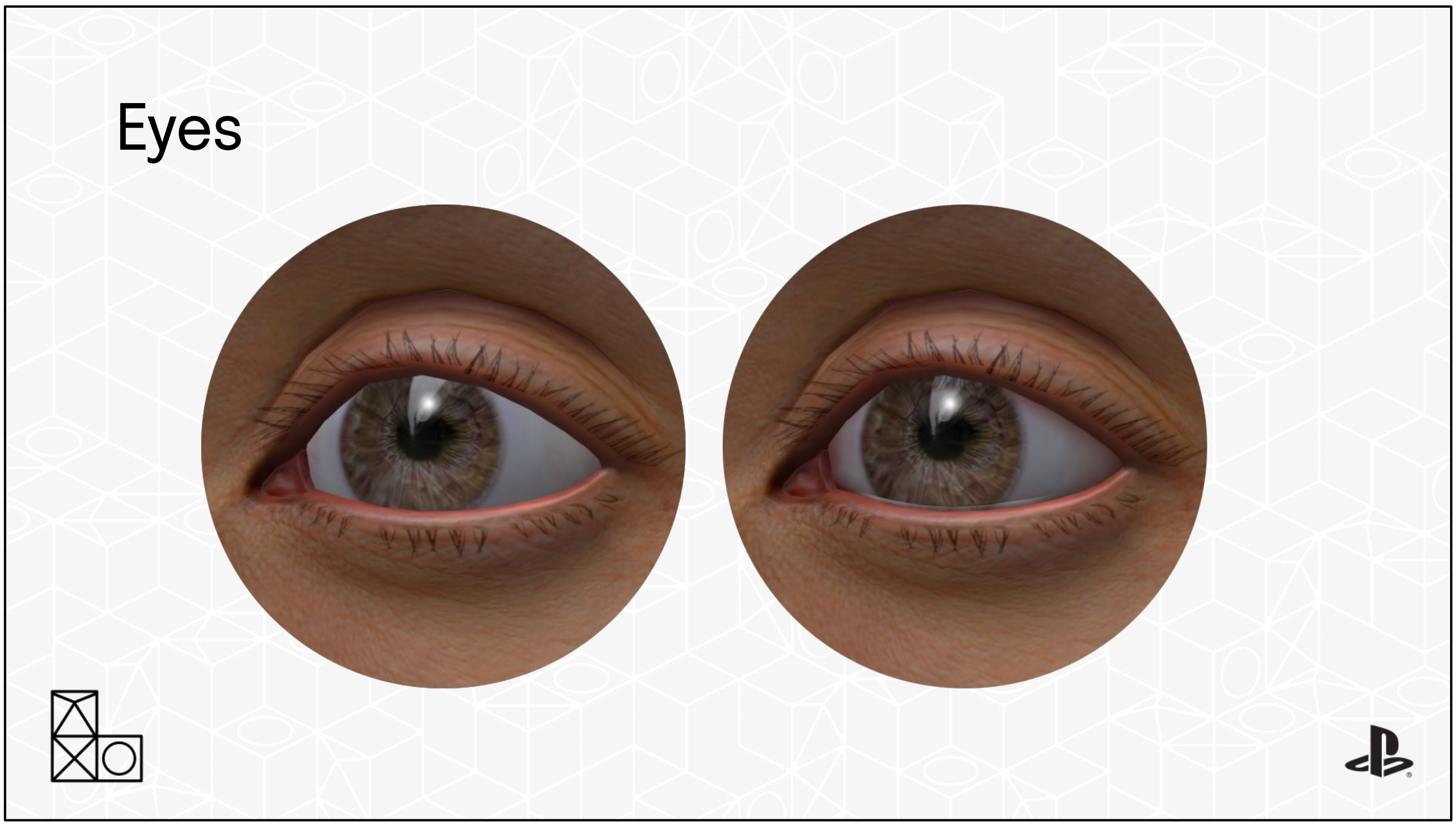
Another nice effect is having a tear line on the lower eyelid. We can add this to the shader by using the bottom set of planes and reversing the normals on the lower part of the eye.



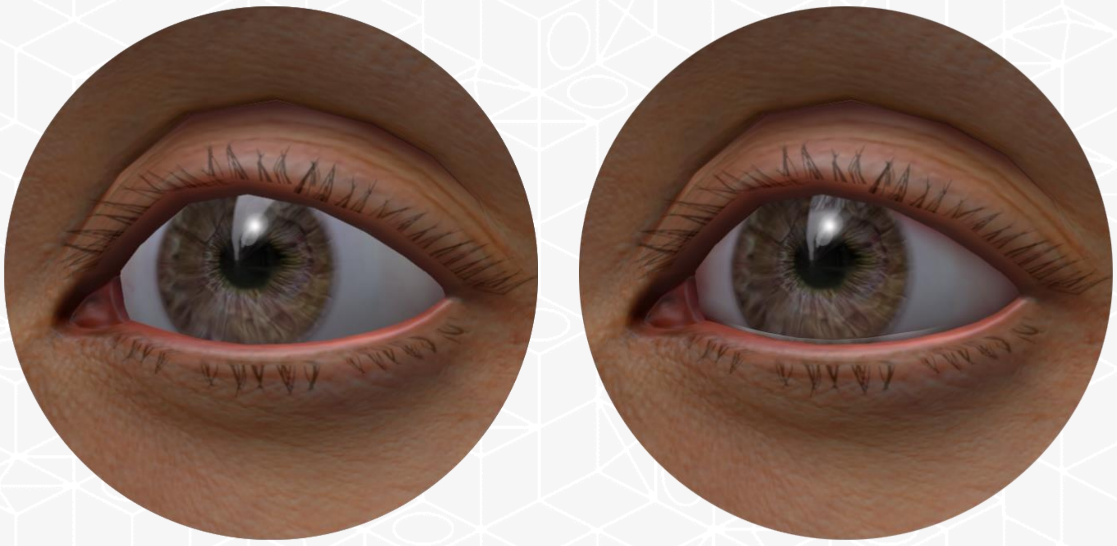
The last thing we can do with the joint positions is approximate eyelash reflections. Based on the eye corner joints and the center of the top lid, we can define a virtual quad in the shader, very much like we defined the iris plane.



And by taking the reflection vector we can intersect with that quad and get a UV coordinate, which we use to look up an eyelash texture. This helps to break up any specular highlights on the top of the eye and will work at any distance.



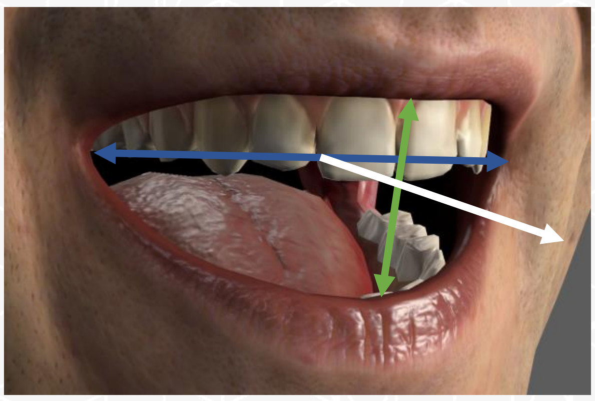
Eyes



The result is an eye that seems much more bedded in, with more interesting colour variation and specular. This is also easier to maintain for the character department, as it doesn't involve any extra geometry skinned to the eyes which may have interpenetration issues.

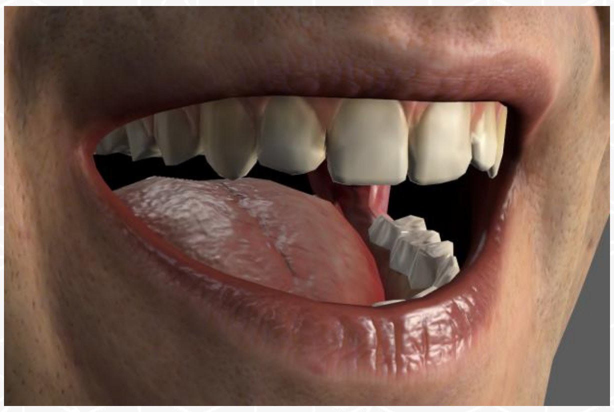
Mouth

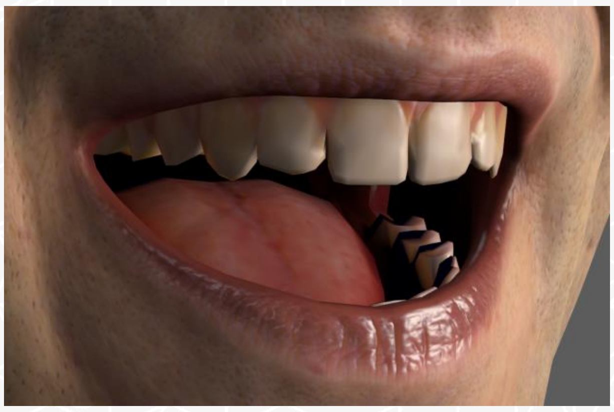
- Similar problem to the eyes
- Use blurred normal like skin
- Use same technique for AO as eyes
- Bent normal based on head forward direction



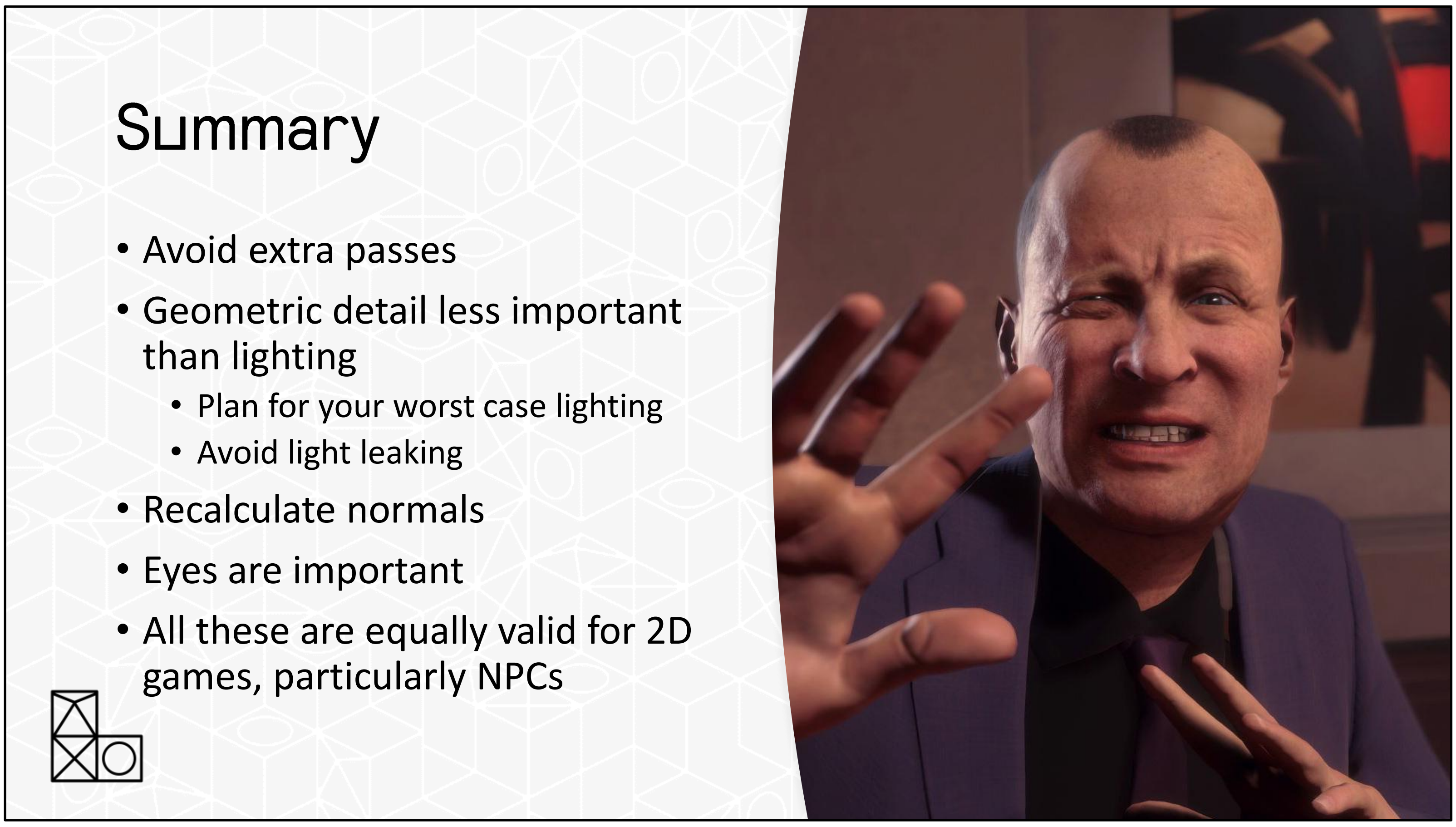
The mouth presents a similar problem to the eyes, it's a common problem in games to have the teeth too brightly lit. We can use the same technique to generate ambient occlusion from the lips on the teeth by choosing 8 bones around the mouth.
On top of this, we would also like to reduce light leaking coming from unlikely directions. We can take a horizontal and vertical vector of the mouth from the joints present, and take the cross product of those to generate an idea of the direction of the open mouth.
We can create a bent normal by biasing the normals that point away towards this direction. It's not entirely correct, but it serves the purpose of stopping light leaking in the mouth.

Mouth



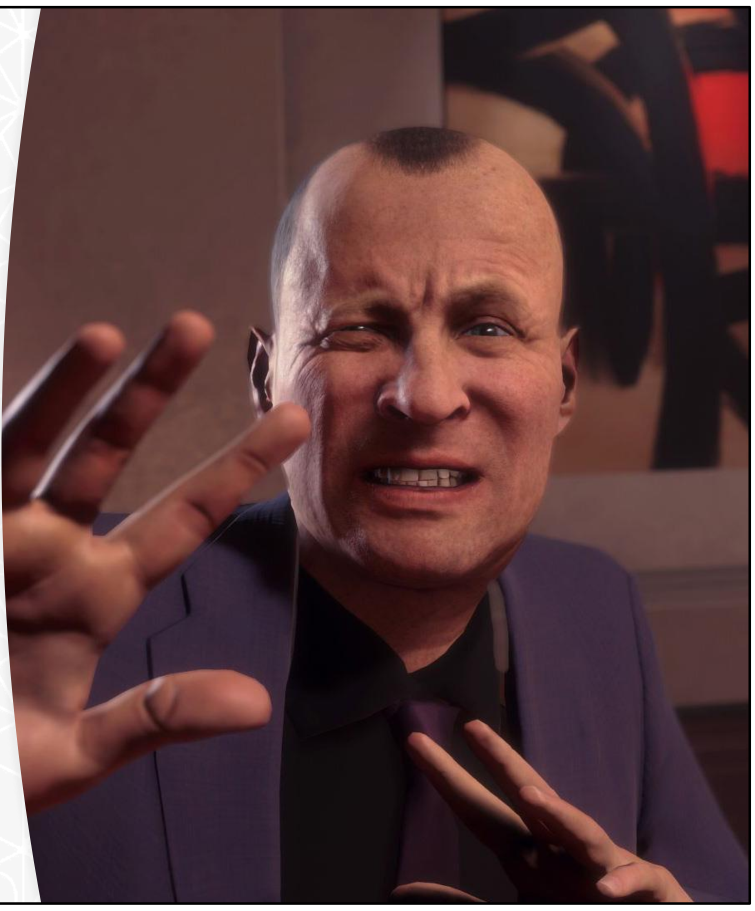


This is the result. The scene here does not have any shadows, and you can see it still does a reasonable job of stopping leaking.



Summary

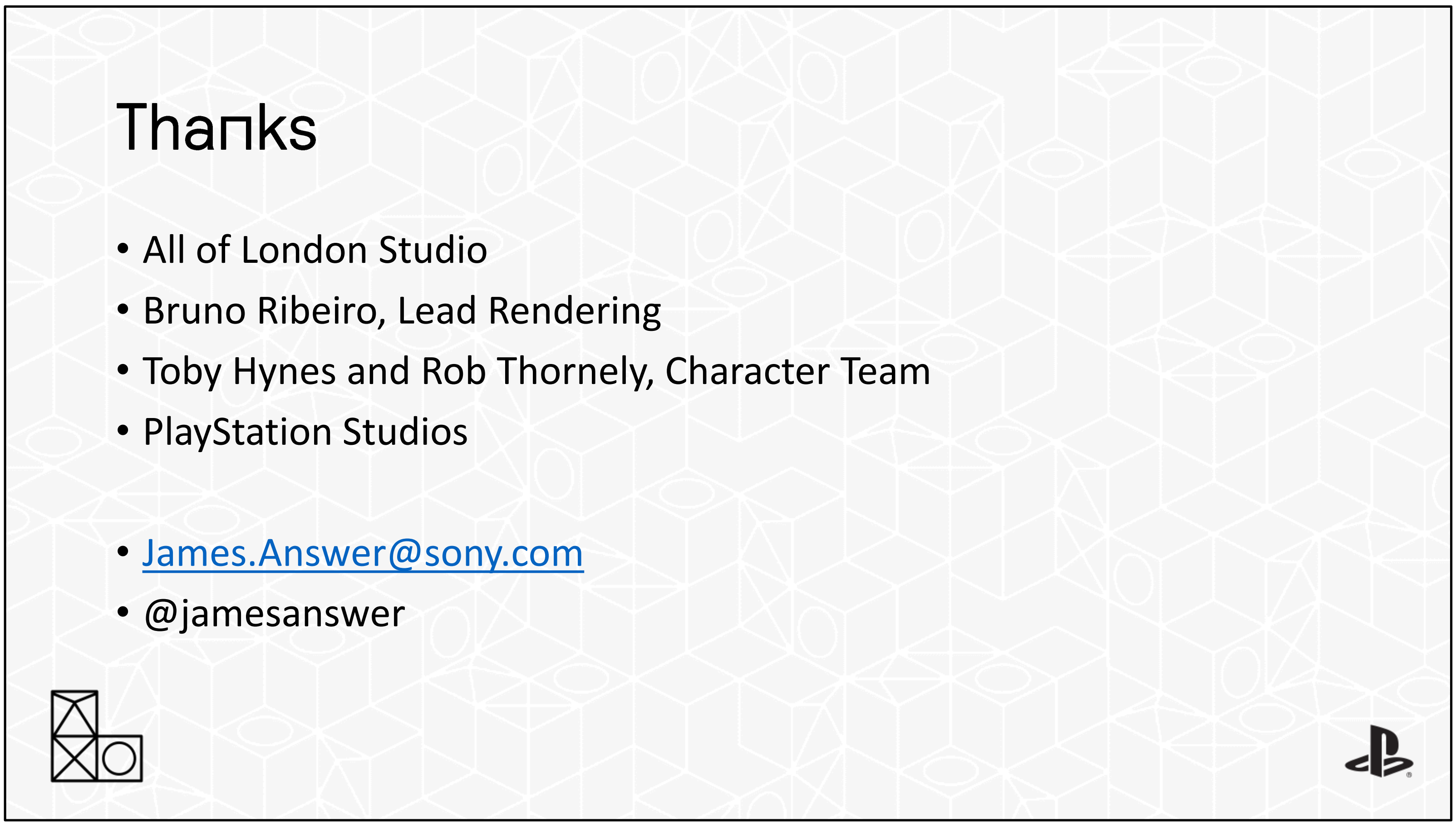
- Avoid extra passes
- Geometric detail less important than lighting
  - Plan for your worst case lighting
  - Avoid light leaking
- Recalculate normals
- Eyes are important
- All these are equally valid for 2D games, particularly NPCs



In summary, for VR to hit higher resolutions it's a good idea to avoid extra passes, so try to use techniques that can all run in a forward shader.

We found that for believable humans that don't have closeups, lighting is far more important than geometric detail.
Spend as much budget as you can in cutscenes on lighting and shadowing, but also be aware that there are situations you can't control, and make sure your lighting model copes with these
Add shadowing terms where possible to avoid light leaking
Make sure you recalculate normals so that the facial performance comes through as intended
Nail the eyes and ensure they read correctly at a distance

Thank you for listening to my talk, I hope you can use it to make your characters look better and run faster.
Many of the concepts in this talk are also applicable to 2D games, particularly NPCs in game where you may not have the budget for screen space subsurface scattering, and where lighting is less tightly controlled.

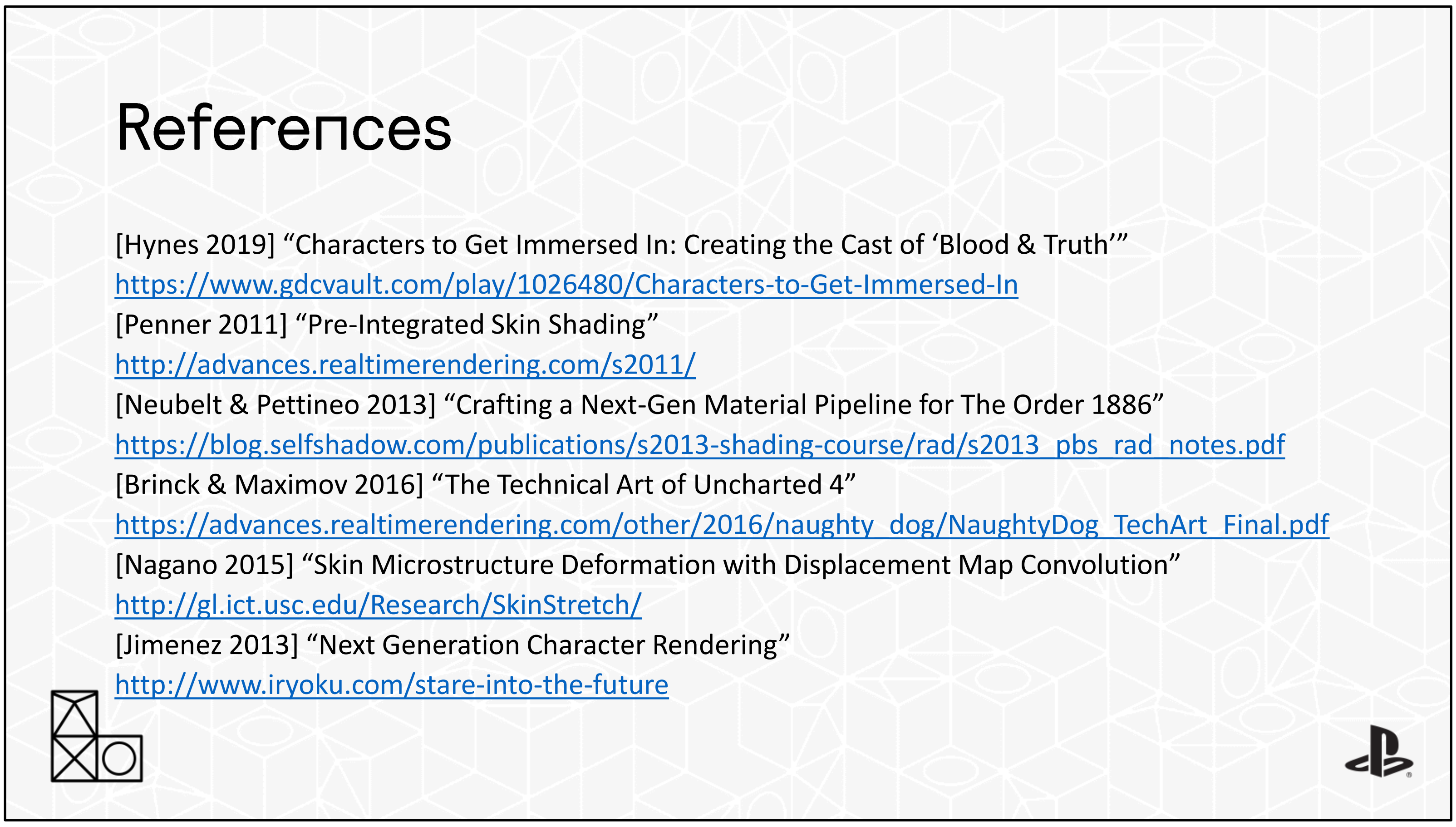


Thanks

- All of London Studio
- Bruno Ribeiro, Lead Rendering
- Toby Hynes and Rob Thornely, Character Team
- PlayStation Studios

- James.Answer@sony.com
- @jamesanswer

I'd like to give my thanks all of London Studio, and particularly to Bruno, our lead rendering engineer, Toby and Rob on the character team, and the rest of PlayStation Studios for their ideas and input and setting the bar in character rendering. This was a short talk with a lot to cover, so if anyone has further questions feel free to reach out to me via e-mail or Twitter.



References

[Hynes 2019] “Characters to Get Immersed In: Creating the Cast of ‘Blood & Truth’”
https://www.gdcvault.com/play/1026480/Characters-to-Get-Immersed-In
[Penner 2011] “Pre-Integrated Skin Shading”
http://advances.realtimerendering.com/s2011/
[Neubelt & Pettineo 2013] “Crafting a Next-Gen Material Pipeline for The Order 1886”
https://blog.selfshadow.com/publications/s2013-shading-course/rad/s2013_pbs_rad_notes.pdf
[Brinck & Maximov 2016] “The Technical Art of Uncharted 4”
https://advances.realtimerendering.com/other/2016/naughty_dog/NaughtyDog_TechArt_Final.pdf
[Nagano 2015] “Skin Microstructure Deformation with Displacement Map Convolution”
http://gl.ict.usc.edu/Research/SkinStretch/
[Jimenez 2013] “Next Generation Character Rendering”
http://www.iryoku.com/stare-into-the-future

Finally here’s a list of the references to papers used in this talk.