Moving to the Next Generation - The Rendering Technology of Ryse

Nicolas Schulz
Senior Rendering Engineer, Crytek

Introduction Ryse

- Launch title for Xbox One
- Close-combat game taking place in ancient Rome
- Story-driven adventure, ~2 hrs of cinematics
- Started in Budapest as Kinect title but full reboot in Frankfurt for next-gen
- A lot of novelties and unknowns for Crytek
  - New genre (3rd person perspective)
  - First console launch title
  - New evolving platform
  - New team
- Small team of rendering engineers fully dedicated to the project

Rendering Challenges

- Dawn of a new console generation
- Ryse designated as visual showcase for Xbox One from beginning
- Target hardware less powerful than high-end PCs at launch
- Major challenge: How can you still get people excited for next-gen visuals?

Rendering Challenges

- Crysis 3 already a visually rich game on Ultra settings
  - Adding just more not an option on weaker hardware
  - Post-processing already maxed out in previous generation
- Focused on consistency of core components instead and worked on improving the details
  - Shading, material definition, lighting quality and global illumination effects
  - Strong focus on what you see all the time as opposed to specific features

Rendering Challenges

- Wanted to get away from the typical “gamey” look
  - No real material definition (mostly due to lack of reflections)
  - Overly high contrast to make flat diffuse materials more visually appealing
  - Noisy image when specular is used (shading aliasing)
  - Over the top usage of post-processing to cover image quality deficiencies
- Wanted to get a step closer to the aesthetics and quality of CG films
  - Well recognizable materials
  - Clean image with little to no aliasing
  - Soft lighting, global illumination effects like light bleeding and natural occlusion
- One key to that is to ground rendering more on physically based paradigms

Physically Based Shading

Physically Based Shading Overview

- Models light-material interaction based on real-world behavior
    - General strong focus on consistency, everything obeys to one well defined rule set
    - Takes a lot of guesswork out of graphics programming
- Considerable implications on several areas
    - Material Model
        - Enforces plausible material parameters and discourages unrealistic setups
        - Defines clear rules for assets, leading to more art/content consistency
    - Shading Model
        - More complex BRDFs, Fresnel, normalization of specular highlights, energy conservation in general
    - Lighting Model
        - **Have to be careful to preserve material integrity through entire pipeline**
        - Real-world reflection ratios useless if light source can randomly add diffuse contribution without affecting specular
- Physically Based Shading can only work well if it gets respected in all areas

Material Model Overview

- Most common attributes
  - Diffuse albedo
    - Not specifically calibrated in Ryse due to time constraints
  - Specular reflectance
    - Based on IOR values
  - Surface roughness
    - Found inverse roughness to be more intuitive to author (smoothness maps)
  - Per-pixel normal
- Special attributes
  - Translucency
  - Subsurface scattering profile

Shading Model

- Unified shading model, expressive enough for 99% of materials
- Specular BRDF
  - Cook-Torrance microfacet model [COOK82]

$$f(l,v)=\frac{D(h)F(v,h)G(l,v,h)}{4(n\cdot l)(n\cdot v)}$$

  - Normal Distribution Function (NDF): GGX [WALTER07]

$$D(h)=\frac{\alpha^2}{\pi((n\cdot h)^2(\alpha^2-1)+1)^2}$$

    - Relatively efficient, has wider tail than other NDFs giving more natural highlights
    - Roughness remapped to generate more perceptually linear highlights
      - More intuitive for artists, also more correct linear filtering

$$\alpha=(1-smoothness*0.7)^6$$

      - Similar distribution as Blinn-Phong power of $2^{smoothness*16}$

Shading Model

- Specular BRDF
  - Fresnel Term: Common Schlick approximation [SCHLICK94]
    - Linear interpolation between specular color F0 and white

$$F(v,h) = F0 + (1 - F0)(1 - (v \cdot h))^5$$

  - Visibility Term
    - Evaluated many terms, opted for Schlick-Smith approximation in the end [SCHLICK94]
    - Using remapped roughness, to avoid highlights getting too hot on smooth surfaces and reduce gain on rough materials at grazing angles (artistic choice)

$$G(l,v,h) = G_1(l)G_1(v) \qquad G_1(v) = \frac{n \cdot v}{(n \cdot v)(1-k)+k} \qquad k = \frac{(0.8+0.5\alpha)^2}{2}$$

Shading Model

- Diffuse BRDF
  - Standard Lambertian BRDF just a flat constant color
    - N.L term is just accounting for increased projected area at grazing angles
    - Assumes surface to be a perfectly isotropic diffusor without any view dependency
  - Oren-Nayar model used in Ryse [OREN94]
    - Takes into account retro-reflection based on surface roughness
      - Subtle but nice quality improvement for rough materials like stone
    - Converges to Lambertian model for smooth materials
    - Efficient approximations for the full quality model exist [FUJII][GOTANDA12]

Deferred Shading

- Started with Crysis 2 codebase which used Deferred Lighting/Light Prepass
  - Minimal GBuffer with just normals and roughness
  - Irradiance and radiance computed in deferred pass
  - Material attributes applied in a second forward pass
- Deferred Lighting does not work well with Physically Based Shading
  - Fresnel term requires specular reflectance (F0) for shading
- Performed transition to full deferred shading for Ryse
  - Bonus: Less draw calls since just one scene pass required
  - Eventually made its way to previous-gen consoles with more aggressive GBuffer packing [SOUSA13]

Deferred Shading GBuffer

- Using 3 tightly packed 32 bit render targets (RGBA8) plus device depth-stencil surface

RT0	RGB: Normals XYZ	A: Translucency Luminance/Prebaked AO Term
RT1	RGB: Diffuse Albedo	A: Subsurface Scattering Profile
RT2	R: Roughness	GBA: Specular YCbCr/Transmittance CbCr

- Normals encoded using BFN approach to avoid 8 bit precision issues [KAPLANYAN10]
- Specular color stored as YCbCr to better support blending to GBuffer (e.g. decals)
  - Allow blending of non-metal decals despite not being able to write alpha during blend ops
  - Can still break when blending colored specular (rare case that was avoided on art side)
- Specular chrominance aliased with transmittance luminance
  - Exploiting mutual exclusivity: colored specular just for metal, translucency just for dielectrics
- Support for prebaked AO value but was just used rarely in the end

Forward+ versus Deferred

- Considered Forward+ at the beginning, in combination with MSAA [MCKEE12]
- Many open challenges in practice
  - How to handle surface modifiers like decals or wetness efficiently
  - Requires two rendering passes again for efficient light culling
  - Most research so far considered just simple light models (mostly point lights)
    - Many more different light types used in practice (projectors, shadow casting lights, area lights, environment probes, etc.)
    - Potentially low wave occupancy due to number of GPRs required for branching when using complex light models
    - Potential overshading/performance waste due to quad occupancy of tiny triangles
  - Definitely still an interesting option for the future though
- Ended up with hybrid approach
  - Majority of objects going through efficient full deferred shading path
  - Forward+ rendering for materials that have very specific shading requirements (mostly hair and eyes)

Specular Aliasing

- Physcially Based Shading very prone to specular aliasing
  - High luminance values from normalized BRDF in combination with high-frequency normal information
- Downsampling normal maps discards information
  - Variance gets reduced when normals are averaged
  - Bumpy normal maps become flat with smaller mips
- Normals and roughness strictly coupled in Ryse
  - Conceptually connected, normals represent surface bumpiness on macro scale, roughness on micro scale
  - Roughness stored in normal map alpha channel in source assets
  - Normal variance of mips baked into roughness maps [HILL12]
    - Toksvig factor used to estimate variance and derive new roughness [TOKSVIG04]

Specular Aliasing

- Problems remain when roughness is modified in GBuffer (decals, rain wetness)
- Addressed by applying normal variance filter in screen space
  - Depth aware filter to avoid issues at silhouette edges (similar to [MITTRING12])
  - Also helps on thin highly reflective geometry
  - Can produce noticeable outline artifacts, especially at grazing light angles
    - Mitigated a bit by reducing specular reflectance for dielectrics

```
specLumLinear *= (finalSmoothness + epsilon) / (originalSmoothness + epsilon);
```

- Found gained temporal stability to be more important than additional artifacts

Specular Aliasing

GBuffer
Filter
Disabled

Specular Aliasing

GBuffer
Filter
Enabled

Light Model

- Common types of analytical light sources used for direct lighting
  - Point lights, projectors, all with support for shadow casting
- No analytical area light model used in Ryse
  - Less of an issue for Ryse since scenario doesn't have artificial light sources (mostly fires instead of light bulbs and tubes)
  - Generally important with PBR to avoid unnatural tiny highlights
  - Just clamped minimum highlight size in BRDF (worked well enough)

Light Model

- Light Attenuation
  - Geometric attenuation accounting for photons spread over larger area at further distance from emitter
  - Traditional radius based attenuation models not very natural $\left(1-\frac{dist}{r}\right)^2$
  - Switched to more physically based model computing radiance emitted from sphere

$$att = \left(1+\frac{dist}{bulbsize}\right)^{-2}$$

  - Bulb size allowed to have unnatural values (e.g. be very high to fake directional sun light)
  - Normalized light intensity, so that specified value is reached 1 meter away from light surface

$$normfactor * \left(1+\frac{1}{bulbsize}\right)^{-2} = 1 \Rightarrow normfactor = \left(1+\frac{1}{bulbsize}\right)^{2}$$

  - Cutoff radius still required for practical reasons (performance, light leaking)
    - Light fades to 0 in last 20% of cutoff radius

Indirect Lighting

- Dropped all constant and hemispherical ambient terms that have just diffuse contribution without any specular
    - Would break the specular reflectance ratios and flatten materials
- Most indirect lighting captured by localized environment probes
- Probes augmented by screen space reflections to get more accurate localization
- Ambient lights to help breaking uniformity

Environment Probes

- Image Based Lighting core feature in CRYENGINE for several years [MITTRING09]
  - Worked on improved consistency with analytical BRDF and usability for Ryse
- Environment probes captured manually at key locations in levels
  - Around 100 probes used per level in Ryse
  - Compressed using BC6H, size 256x256 for specular cubemaps
- Cubemaps preconvolved offline using custom version of ATI CubemapGen
- Probes sorted by locality based on layers
  - Local probes overwrite more global probes

Environment Probes

- Parallax correction using geometry proxies to obtain better locality of reflections [BJORKE][LAGARDE12]
- Smooth blend weight falloff to avoid harsh transition between probes
  - Probes are oriented boxes to better fit indoor areas
  - Blend weight for box computed by mapping cube to sphere before applying attenuation function [NOWELL05]

```
float3 MapCubeToSphere( float3 pos )
{
	float3 posSq = pos.xyz * pos.xyz;
	return pos * sqrt( 1 - 0.5 * posSq.yzx - 0.5 * posSq.zxy + 0.333 * posSq.yzx * posSq.zxy );
}
```

Ambient Lights

- Probes stretched over a larger area can lack local light intensity changes resulting in flat ambient
  - Artists need a way to more accurately set up bounce lighting
  - Using regular lights creates undesired specular highlights
  - Using just diffuse from regular lights would break material integrity and flatten surfaces
- Our solution: Ambient Lights
  - Essentially multiplicative light sources applied on top of probes
  - Affect diffuse and specular equally and maintain reflectance ratio
  - Intensity > 1 used to add bounce lighting
  - Intensity < 1 used to darken ambient and emulate ambient occlusion

Ambient Lights

Disabled

Ambient Lights

Enabled

Glossy Realtime Local Reflections

- Locality of probes still very limited (even with parallax correction)
  - Can't capture small-scale reflections or reflections from dynamic objects
- Exploit image space information where available
  - Screen Space Reflections
  - SSR first used for Crysis 2 DX11 [SCHULZ11]
  - Further evolved SSR for Ryse to work with different material roughness levels

Glossy Realtime Local Reflections

- Possible solution: 2.5D cone tracing, similar to Voxel Cone Tracing
  - Use preconvolved scene buffer and min/max depth hierarchy while tracing along reflection vector
  - Cone width depending on roughness and ray distance determines which mip level to sample
- Opted for simpler and slightly cheaper solution for Ryse
  - Perform simple raytracing step to get mirror reflections
  - Build convolved versions of mirror reflections by repeated downsampling and Gaussian filtering
  - Alpha is convolved as well, making unsharp reflections less visible
  - Material roughness determines which mip level to blend on top of probe reflections
  - Less correct than cone tracing, does not account for increased blurriness in distance
  - Little aliasing for rougher materials since convolution is done after tracing (essentially applying a low pass filter)

Facial Rendering

Facial Rendering

- Facial animation and rendering large focus since beginning
  - Ryse very story-driven game with huge amount of cinematics
  - Raising quality bar for characters compared to previous generation was one of the designated project goals
- Largely relying on general shading and lighting improvements
- Some specific solutions required for facial features

Skin Rendering

- Skin goes through unified shading model
  - Standard BRDF in Ryse is more advanced than what was used for the NVidia Human Head demo [EON]
  - GGX with wide tail works fairly well for specular highlights
    - No urgent need for using multiple lobes to get smoother highlights
- Subsurface Scattering
  - Unified subsurface scattering solution
    - Also used for marble
  - Optimized for skin
    - Many close-up shots on faces
    - Errors in skin scattering profile more noticeable

Subsurface Scattering

- Screen-space convolution applied to irradiance buffer after lighting passes
  - Take into account perspective distortion, FOV and aspect ratio to maintain fixed world-space scattering radius
- Convolution based on a pseudo-separable cross bilateral filter [MIKKELSEN10]
  - Two Gaussians combined to determine final kernel weights for convolution
  - First Gaussian with different weights for each color channel (scattering radius for red higher than for green and blue)
  - Second Gaussian takes into account depth differences
- Single Gaussian not enough to approximate skin scattering profile [EON]
  - Scattering profile consists of a sharp spike and a broad base
  - Results will look either blurry or too sharp
  - Blend between original irradiance and Gaussian to approximate spike in profile

Subsurface Scattering

Disabled

Subsurface Scattering

Enabled

Skin Translucency

- Light bleeding through ears and nostrils
- Mostly relying on unified solution shared with vegetation
  - Inverted normal used to estimate amount of light entering from the backside
  - Artists specify density/thickness using translucency map
  - Support for transmittance filter color (wavelength dependent extinction)

- Transmittance color for skin computed from thickness to approximate natural gradient

```
float3 transmittanceColor = exp( (1 - fTranslucency) * float3( -8, -40, -64 ) );
```

Hair Rendering

- Hair shading well researched topic
- Standard model is Marschner that is commonly used in high quality offline rendering [MARSCHNER03]
  - Hair exposes 2 highlights shifted along hair strands
  - Primary highlight due to specular reflection (hence monochrome)
  - Secondary highlight due to light transmission (partly absorbed hence colored)
- Cheaper approximation by generating 2 highlights using Kajiya-Kay model [SCHEUERMANN04]
- Both models work well, we opted for Kajiya-Kay since hair rendering can be very performance-heavy due to overdraw
- Direction map specifying hair tangent essential for quality

Hair Rendering

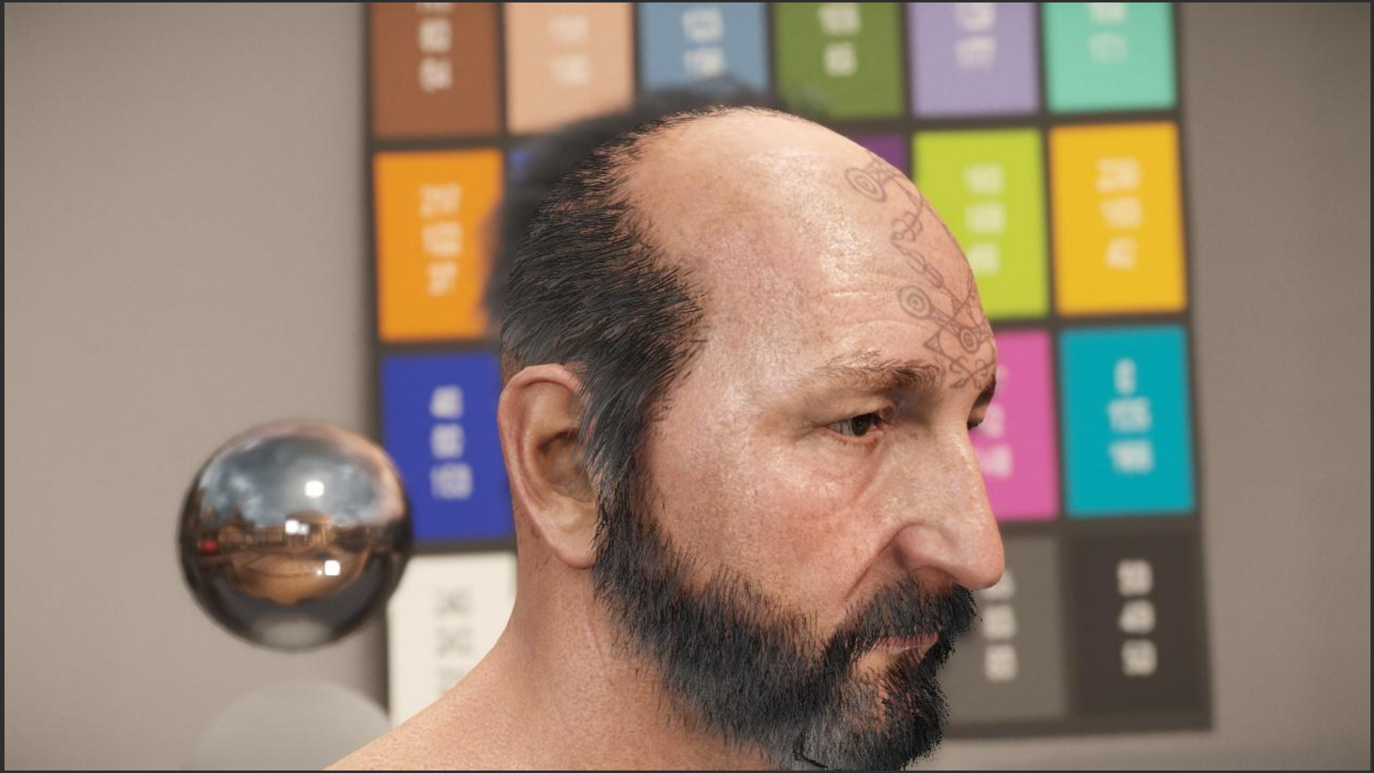
- Main challenge is how to avoid aliasing and make hair look smooth
  - Particularly true for thin individual facial hair as in beards
  - Alpha tested hair can look wiry and is temporally very unstable
  - Alpha blended hair smooth but exposes the well-known shortcomings
    - Sorting issues without order independent transparency
    - Requires forward shading
    - Issues with deferred passes and post processing (mostly DOF) since no depth is written
  - Most high-end solutions so far rely on high MSAA sample counts [MITTRING11][ZIOMA12]
    - Not feasible for realtime usage on consoles
- Tried many variations, rendering more opaque parts to GBuffer, more transparent tips alpha-blended
  - Never got the desired look for some hair types
  - Only fully alpha blended hair provided the quality we wanted for certain types of beards

Hair Rendering

- New "thin hair" feature, specifically for hair meshes that are not dense and where individual hair strands visible
  - Fully alpha blended for smoothness
  - Sorting issues need to be addressed on art side
    - Hair directly on top of skin usually doesn't have sorting issues since skin acts as occluder
    - Using mixture of alpha tested caps and thin hair for more complex hair meshes
  - Shading applied using light list generated during tiled shading (Forward+ style)
  - Post-processing issues addressed by "depth fixup" pass
    - Hair does not write device depth to avoid self-occlusion issues
    - Write approximate (alpha-tested) depth to alpha channel during color blending pass
    - Final merge pass combines alpha depth values with original depth before post processing
    - Alpha testing issues (aliasing, thickness) much less noticeable in post processing effects
    - Also used for some particle effects to make them work with DOF

Depth Fixup

Disabled



Depth Fixup

Enabled

Performance

Rendering Resolution

- Ryse performs scene rendering in 900p
  - Sweet spot: Quality per pixel versus number of pixels
- Swapchain backbuffer is 1080p
  - Text very prone to show upscaling artifacts
  - All in-game UI and menus rendered in FullHD 1080p
- Scene gets upscaled after rendering
  - Using custom upscaling pass
    - Smoothstep function applied to fractional part of pixel coordinates
    - Increased sharpness due to mixture of nearest neighbor and linear filtering
    - Efficient: just a single bilinear sample and a few ALUs on the texture coordinates
  - Did not yet evaluate the Xbox hardware upscaler

Tiled Deferred Lighting

- Deferred shading heavy on bandwidth usage/memory traffic
  - Overlapping lights cause considerable amount of redundant read/write operations
- Tiled shading [ANDERSSON11]
  - Split screen into tiles and generate a list of all lights affecting each tile in CS
  - Cull lights by min/max depth extents of tile
  - Loop over light list for each tile and apply shading
  - Great bandwidth savings due to just reading GBuffer once and writing shading results once at the end for each pixel
- Single compute shader in Ryse for light culling and executing entire lighting/shading pipeline

Tiled Deferred Lighting

- Challenges
  - Frustum primitive culling not accurate, creates false positives
    - Often considerably more pixels shaded than with stencil tested light volumes
  - Handling light resources (all resources need to be accessible from CS)
    - Shadow maps stored in large atlas
    - Diffuse and specular probe cubemaps stored in texture arrays
    - Projector textures stored in texture array (have to use standardized dimensions and format)
  - Keeping GPRs under control
    - Dynamic branching for different light types
    - Deep branching requires additional GPRs and lowers occupancy
    - Had to manually rearrange code to stay within desired GPR limit
- Average savings: 2 – 5 ms, in worst case scenarios a lot more

Shadow Map Optimization

- Static shadow map
  - Takes advantage of increased memory on new platform
  - Generate large shadow map with all static objects once at level load time
    - Or when transitioning into a different area
  - Static map used as replacement for 4th and 5th shadow cascade
  - Avoids re-rendering distant static objects every frame
  - 8192x8192 16 bit shadow map (128 MB) covering a 1km area of the game world provides sufficient resolution
  - Saved 40%-60% of drawcalls in shadow map passes

Thanks for Your Attention

Special Thanks

Theodor Mader and Jerome Charles from the Ryse Rendering Team

Tiago Sousa, Stephen Clement, Axel Gneiting, Bogdan Coroi, Carsten Wenzel, Chris Raine, Chris Bolte, Minghao Pan, Chris Campbell, Ats Kurvet, Abdenour Bachir, Florian Reschenhofer, Hayo Koekoek, Hanno Hagedorn, Chris Evans and many more...

Baldur Karlsson for RenderDoc
http://cryengine.com/renderdoc

We are looking for more talent to join our team... ☺
http://www.crytek.com/career/offers/overview/frankfurt/programming-engineering

References

[ANDERSSON11] Johan Andersson, "DirectX 11 Rendering in Battlefield 3", GDC 2011
[BJORKE] Kevin Bjorke, "Image-Based Lighting", GPU Gems 1
[COOK82] Robert L. Cook, and Kenneth E. Torrance, "A Reflectance Model for Computer Graphics", ACM Transactions on Graphics 1982
[EON] Eugene d'Eon, and David Luebke, „Advanced Techniques for Realistic Real-Time Skin Rendering", GPU Gems 3
[FUJII] Yasuhiro Fujii, "A tiny improvement of Oren-Nayar reflectance model", http://mimosa-pudica.net/improved-oren-nayar.html
[GOTANDA12] Yoshiharu Gotanda, „Beyond a Simple Physically Based Blinn-Phong Model in Real-Time", Siggraph Shading Course 2012"
[HILL12] Stephen Hill and Dan Baker, "Rock-Solid Shading", Siggraph 2012
[JIMENEZ12] Jorge Jimenez, "Separable Subsurface Scattering and Photorealistic Eyes Rendering", Siggraph 2012
[KAPLANYAN10] Anton Kaplanyan, "CryENGINE 3: reaching the speed of light", Siggraph 2010
[KARIS13] Brian Karis, "Real Shading in Unreal Engine 4", Siggraph Shading Course 2013
[LAGARDE12] Sébastien Lagarde, and Antoine Zanuttini, "Local Image-based Lighting with Parallax-corrected Cubemap", Siggraph 2012
[LAZAROV13] Dimitar Lazarov, "Getting More Physical in Call of Duty: Black Ops II", Siggraph Shading Course 2013
[MARSCHNER03] Stephen R. Marschner, Henrik Wann Jensen, Mike Cammarano, Steve Worley, and Pat Hanrahan, "Light scattering from human hair fibers", ACM Trans. Graph. 22
[MCKEE12] Jay McKee, „Technology behind AMD'S Leo Demo", GDC 2012
[MIKKELSEN10] Morten Mikkelsen, "Skin Rendering by Pseudo-Separable Cross Bilateral Filtering", 2010, https://dl.dropbox.com/u/55891920/papers/cbf_skin.pdf
[MITTRING09] Martin Mittring, "A bit more deferred – CryEngine3", Triangle Game Conference 2009
[MITTRING11] Martin Mittring, "The Technology Behind the DirectX 11 Unreal Engine Samaritan Demo", GDC 2011
[MITTRING12] Martin Mittring, "The Technology Behind the Unreal Engine 4 Elemental demo", Siggraph 2012
[NOWELL05] Philip Nowell, "Mapping a Cube to a Sphere", http://mathproofs.blogspot.de/2005/07/mapping-cube-to-sphere.html

References

[OREN94] M. Oren, and S.K. Nayar "Generalization of Lambert's Reflectance Model", Siggraph 1994

[SCHEUERMANN04] Thorsten Scheuermann, "Practical Real-Time Hair Rendering and Shading", Siggraph 2004

[SCHLICK94] Christophe Schlick, "An Inexpensive BRDF Model for Physically-based Rendering", Computer Graphics Forum 1994

[SCHULZ11] Tiago Sousa, Nicolas Schulz, and Nick Kasyan, "Secrets of CryENGINE 3 Graphics Technology", Siggraph 2011 http://www.crytek.com/cryengine/presentations

[SOUSA13] Tiago Sousa, Carsten Wenzel and Chris Raine, "The Rendering Technologies of Crysis 3", GDC 2013, http://www.crytek.com/cryengine/presentations

[SOUSA13-2] Tiago Sousa, "CryENGINE 3 Graphic Gems", Siggraph 2013 http://www.crytek.com/cryengine/presentations

[TOKSVIG04] Michael Toksvig, "Mipmapping Normal Maps", 2004, ftp://download.nvidia.com/developer/Papers/Mipmapping_Normal_Maps.pdf

[WALTER07] Bruce Walter, Stephen R. Marschner, Hongsong Li, and Kenneth E. Torrance. "Microfacet models for refraction through rough surfaces", EGSR 2007

[ZIOMA12] Renaldas Zioma, and Simon Green, "Mastering DirectX 11 with Unity", GDC 2012

Bonus Slides

Environment BRDF

- Standard Fresnel equation only valid for perfectly smooth surfaces
  - Works for analytical BRDF because microfacets are perfect mirrors
  - Not directly applicable to preconvolved radiance environment maps
- Need to integrate the complete BRDF over the specular lobe
  - Factor out the radiance that is stored in the cubemap from the integral
  - Leaves a second integral containing the Environment BRDF [LAZAROV13][KARIS13]
- Solution for Ryse similar to [LAZAROV13]
  - Store numerically integrated Environment BRDF for reflectance 0% and 100% in 2D lookup table
    - Parametrized by roughness and N.V
  - Use surface reflectance to interpolate between precomputed values during shading

Eye Rendering

- Eyes essential for believable characters
- Eye composed of 3 major components
  - Eyeball (moves independently)
  - Ambient occlusion layer (moves with eye lids)
  - Specular overlay for tear fluid (moves with eye lids)
- Various detail features to make eyes more believable
  - Lens distortion, iris self-shadowing, subsurface scattering approximations
- Occlusion essential for quality
  - Screen space solutions not reliable enough here
  - Separate geometry layer with hand-painted maps for ambient and reflection occlusion

Eye Rendering

- Light scattering by lens/cornea important feature for making eyes more expressive
  - Accurate solution: using 3D LUT, prebaked using photon mapping [JIMENEZ12]
  - Very simple and coarse approximation in Ryse to get some of the desired effect
    - Generate concave version of cornea/lens normal map and dot with light vector

```
float3 vConcavityNormalTS;
vConcavityNormalTS.xy = -FetchNormalMap( corneaNormalSampler, baseTC.xy ).xy * ConcavityScale;
vConcavityNormalTS = DecodeCompressedNormal( vConcavityNormalTS.xy );

float scattering = 0.5 + saturate( pow( dot( vLightTS, vConcavityNormalTS ), 8 ) ) * 2;
```