What motivated us to calibrate our lighting and materials on Far Cry 3? Far Cry 3 is a huge game, which puts a lot of pressure on content creation, a pressure which is only going to increase as hardware improves, consumers’ continue to demand better quality and we wish to create bigger and bigger games. With a time of day cycle and indoor and outdoor environments, we need to make sure that our art looks good and holds up in all situations, and also to make sure it’s easy to achieve that goal. We need to create better tools, improve our material models, expose better parameters, link things that should be linked…
On Far Cry 3, we began to make these changes to help our artists create high quality visuals, looking at our shading, our lighting but also our materials.
Getting our diffuse albedo right was very important to us. If you just look at the concept art, you can see bright and saturated colours, which are really, really easy to get wrong. (Hence how many games look rather desaturated, and grey and brown.)

Moreover, balancing diffuse albedo textures causes a lot of problems during development. Maybe at one time or another, you’ve heard the phrase “My characters don’t sit in the scene.” Your character modellers and your environment artists are two separate teams, creating materials to different standards. Or an artist creates a material that looks fine outside, but for some reason looks pitch black as soon as it’s moved to an indoor environment. Of course, he first goes to the lighters to complain, and there’s a horrible cycle of iteration between lighters and material artists trying to work out who’s to blame (and let’s not get started on the postprocessing effects).
So what do we mean by diffuse albedo? Well, it’s part of our BRDF equation…

\[ f(l, v) = \frac{c_{\text{diff}}}{\pi} + \frac{F(l, h)G(l, v, h)D(h)}{4(n \cdot l)(n \cdot v)} \]
This BRDF is a combination of the Torrance-Sparrow microfacet BRDF for specular reflectance...

\[
f(l, v) = \frac{c_{\text{diff}}}{\pi} + \frac{F(l, h) G(l, v, h) D(h)}{4(n \cdot l)(n \cdot v)}
\]
Lambert diffuse BRDF:

\[ f(l, v) = \frac{c_{\text{diff}}}{\pi} + \frac{F(l, h)G(l, v, h)D(h)}{4(n \cdot l)(n \cdot v)} \]

...and Lambert diffuse BRDF. (Remember the N.L term is part of the reflectance integral not the BRDF.) This is where our diffuse albedo comes in, and we can see that it’s a physical property and can be measured. If we get it wrong, this whole equation will be imbalanced, so it doesn’t make much sense to have a physically-based specular reflectance model without sorting out your diffuse albedo at the same time.
Motivation

- Natural, saturated colours
- Consistent across materials
- Consistent under all lighting conditions
- Diffuse lighting balanced with specular

Let’s sum up our motivation.
Diffuse Albedo vs Diffuse Albedo
Diffuse Reflectance vs Specular Reflectance

Too Dark  Correct  Too Bright
If we assume only diffuse lighting here.
After Colour Correction
Capturing Diffuse Albedo

- Use the Macbeth ColorChecker™ as reference.
- 24 colour patches with known sRGB values.
It’s obviously really important to have consistent lighting across the material that you’re capturing and the ColorChecker™ itself. You also want to minimise specular reflection. This means they both should be parallel to the camera plane, with direct lighting minimised as much as possible. Overcast days are ideal.
Many thanks to Paul Malin of Activision Central Tech, for sharing with me his colour correction algorithm and allowing me to present it.
Polynomial Transform [Malin11]

- **Pro**: accurately adjusts levels
- **Con**: channel independent

\[ \forall i \in \{r, g, b\}, \ y_i = \sum_{0}^{m} a_m x_i^m \]
Thus to find the best approximate affine and polynomial transforms, we need to find appropriate $X$ and $Y$, and then find the matrix inverse. Thankfully, we can do this relatively simply by Gauss-Jordan elimination.
- Let $x_j$ be our photographed ColorChecker™ patches.
- Let $y_j$ be the target ColorChecker™ values.

\[
\begin{bmatrix}
  x_{0,r} & x_{0,g} & x_{0,b} & 1 \\
  x_{1,r} & x_{1,g} & x_{1,b} & 1 \\
  \vdots & \vdots & \vdots & \vdots \\
  x_{n,r} & x_{n,g} & x_{n,b} & 1 \\
\end{bmatrix}
\begin{bmatrix}
  a_{11} & a_{12} & a_{13} \\
  a_{21} & a_{22} & a_{23} \\
  a_{31} & a_{32} & a_{33} \\
  a_{41} & a_{42} & a_{43} \\
\end{bmatrix}
= 
\begin{bmatrix}
  y_{0,r} & y_{0,g} & y_{0,b} \\
  y_{1,r} & y_{1,g} & y_{1,b} \\
  \vdots & \vdots & \vdots \\
  y_{n,r} & y_{n,g} & y_{n,b} \\
\end{bmatrix}
\]
Polynomial Transform

- Let $x_j$ be our photographed ColorChecker™ patches.
- Let $y_j$ be the target ColorChecker™ values.

$$\forall i \in \{r, g, b\}, \begin{bmatrix} 1 & x_{0,i} & x_{0,i}^2 & \cdots & x_{0,i}^m \\ 1 & x_{1,i} & x_{1,i}^2 & \cdots & x_{1,i}^m \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n,i} & x_{n,i}^2 & \cdots & x_{n,i}^m \end{bmatrix} \begin{bmatrix} a_{0,i} \\ a_{1,i} \\ a_{2,i} \\ \vdots \\ a_{m,i} \end{bmatrix} = \begin{bmatrix} y_{0,i} \\ y_{1,i} \\ y_{2,i} \\ \vdots \\ y_{n,i} \end{bmatrix}$$
Colour Correction Tool

- Command line tool.
- Launched by a Photoshop script.
- Operates in xyY colour space.
- Applies the following transforms [Malin11]:

Affine → Polynomial → Affine
In Photoshop we first have to click around the ColorChecker to tell the tool where to find it.
Then we can run the colour calibration script.
Future Work

Improve tool:
- ColorChecker™ detection
- dcraw to obtain raw sensor data
- Adobe Camera Model for lens correction

Improve capturing:
- Investigate polarising filters.
After Lens and Colour Correction
Lighting
Sky Colour

- Hue and saturation given by two gradient textures:
  - Sun side
  - Sun opposite side

- Luminance given by CIE Sky Model.
- Cloud layers blended on top.
CIE Sky Model

- Models the \textit{luminance} distribution of the sky.
- Based on angular distances:
  - $\varphi$ - sky element to zenith.
  - $\theta$ - sky element to sun.
  - $\theta_z$ - sun to zenith.

This only models the luminance of the sky, not its hue or saturation.
The coefficients can change the model from a clear sky model to an overcast sky model, as well as differentiating between different atmospheric conditions.

Luminance is given relative to that of the zenith:

\[
\frac{L_{\theta,\phi}}{L_{\theta_z}} = \frac{(1 + a \exp(\frac{b}{\cos \phi}))(1 + c(\exp(d\theta) - \exp(d\pi/2) + e \cos^2 \theta))}{(1 + a \exp(b))(1 + c(\exp(d\theta_z) - \exp(d\pi/2)) + e \cos^2 \theta_z)}
\]

Five coefficients to give different sky models.

- \(\varphi\) - sky element to zenith
- \(\theta\) - sky element to sun
- \(\theta_z\) - sun to zenith

The coefficients can change the model from a clear sky model to an overcast sky model, as well as differentiating between different atmospheric conditions.
At dawn and dusk, the intensity at the sun can be 30 times that of the zenith. Thus if you make everything relative to the zenith, overall your sky will get much, much brighter at sunrise and sunset. By making everything relative to the sun intensity, you remove this problem, and in fact, we didn’t have to adjust the sky intensity at all.

CIE Sky Model

- Make relative to sun intensity not zenith intensity:
  - More consistent sky luminance over all times of day

\[
\frac{L_{\theta,\phi}}{L_{\theta_z}} = \frac{(1 + a \exp(b/\cos \phi))(1 + c(\exp(d\theta) - \exp(d\pi/2)) + e \cos^2 \theta)}{(1 + a \exp(b/\cos \theta_z))(1 + c(1 - \exp(d\pi/2)) + e)}
\]

- \(\phi\) - sky element to zenith
- \(\theta\) - sky element to sun
- \(\theta_z\) - sun to zenith
CIE Sky Model

- Chose custom coefficients:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIE clear sky model</td>
<td>-1.0</td>
<td>-0.32</td>
<td>-10.0</td>
<td>-3.0</td>
<td>0.45</td>
</tr>
<tr>
<td>Far Cry 3 sky model</td>
<td>-1.0</td>
<td>-0.08</td>
<td>-24.0</td>
<td>-3.0</td>
<td>0.30</td>
</tr>
</tbody>
</table>
Sky Lighting

- Use sky dome as hemispherical light.
- Create SH coefficients from gradients and CIE model.
- Sky lighting and sky will always match.
- Variations in sky lighting.

This ensures that sky lighting and the sky itself will always match up.
Original Light Probe

Midday
Final Light Probes

Dawn  Morning  Afternoon  Dusk
Artists obviously found the loss of control difficult, especially when trying to replicate concept art. They often wanted to achieve a mix of colours, so one colour for the ambient and another for the sky, to achieve the desired artistic goal.
Polarising Filter

Without a polarising filter  With a polarising filter

PiccoloNamek at the English language Wikipedia
Polarising Filter

- Apply a fake polarising filter in the sky dome shader:
  \[ \text{skyColor}.rgb \times= \text{g\_SkyPostProcess}.rgb \]

- Also reduces direct sky reflection:
  - Applied to the sky in captured cube maps.
Here, we very much follow from Dimitar Lazarov’s presentation at SIGGRAPH 2011 on Physically-Based Rendering in Call of Duty: Black Ops. This is just the specular part, we keep Lambertian diffuse on top of this. This model is great because you can pick and choose distribution and geometric terms to use.

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

- Torrance-Sparrow microfacet BRDF: [Lazarov11]
Use normalised Blinn-Phong:

\[
\frac{1}{4} D(h) = \frac{m + 2}{8\pi} (n \cdot h)^m
\]

- Specular power \( m \) in range \([1, 8192]\)
- Encoded as glossiness \( g \) in \([0, 1]\) where: \( m = 2^{13g} \)
Schlick’s approximation for Fresnel:

\[ F_{Schlick}(v, h) = c_{spec} + (1 - c_{spec})(1 - h \cdot v)^5 \]

Spherical gaussian approximation: [Lagarde12]

\[ F_{SG}(v, h) = c_{spec} + (1 - c_{spec})e^{-6h \cdot v} \]
 Visibility Term

- Call the remaining terms the visibility term:

\[ V(l, v, h) = \frac{G(l, v, h)}{(n \cdot l)(n \cdot v)} \]

- Many games have \( V(l, v, h) = 1 \) making specular too dark. [Hoffman10] [Lazarov11]
Visibility Term

- Schlick-Smith:
  - Roughness dependent.

\[ V(l, v, h) = \frac{1}{(n \cdot l)(1 - a) + a((n \cdot v)(1 - a) + a)} \]

- Calculate \( a \) from Beckmann roughness \( k \) or Blinn-Phong specular power \( m \): [Lazarov11]

\[ a = \sqrt{\frac{2}{\pi}} k = \frac{1}{\sqrt{\frac{\pi}{4} m + \frac{\pi}{2}}} \]
Cost of Schlick-Smith is \textit{8 instructions} per light. [Lazarov2011]

Too expensive for our budget:
- Especially when applied to every light.

Approximate by integrating Schlick-Smith over the hemisphere.
\[
\frac{1}{2\pi} \int_\Omega \frac{1}{((\mathbf{n} \cdot \mathbf{l})(1 - a) + a)((\mathbf{n} \cdot \mathbf{v})(1 - a) + a)} d\Omega \\
= \left[ \int_0^{\frac{\pi}{2}} \frac{\sin(\theta)}{(\cos(\theta)(1 - a) + a)} d\theta \right]^2 \\
= \left[ -\log(a) \right]^2 \left[ \frac{1}{1 - a} \right]
\]
Approximating Schlick-Smith

- Still 8 instructions! 😎
- Dependent only on glossiness…
- …just like our existing energy conservation term.
- Find a cheap curve to approximate both:

\[
E(g) = \frac{2^{13g} + 2}{8} \left[ -\log\left( \sqrt{\frac{1}{4} 2^{13g} + \frac{\pi}{2}} \right) \right]^2 \left[ 1 - \sqrt{\frac{1}{4} 2^{13g} + \frac{\pi}{2}} \right]
\]

Approximate relative to glossiness g as it’s very close to a linear representation of highlight size.
We looked at functions of the form \((am + b)/8\), just a simple modification of our initial energy conservation term (in yellow at the bottom). Observe that we’re careful to keep the value when \(m = 1\) the same. This is because at low roughnesses, our artists desired to keep specular as dull as possible, and any brightening was incredibly noticeable.
Only the yellow and green lines stay beneath the red curve at all points.
As this is an approximation, it both overestimates and underestimates the correct values, and any overestimation was really noticeable.

Also, despite our best efforts to calibrate our diffuse albedo, it erred on the dark side, so our specular was naturally brighter because of that.
Specular Filtering

- Needed specular filtering to:
  - Reduce shimmering.
  - Preserve appearance in the distance.

Without Filtering

With Filtering
Specular Filtering

- Needed specular filtering to:
  - Reduce shimmering.
  - Preserve appearance in the distance.

Without Filtering  With Filtering
Specular Filtering

- Use Toksvig’s formula to scale the specular power, using the length of the filtered normal: [Hill11]

\[ m' = \frac{\| n_a \| m}{\| n_a \| + m(1 - \| n_a \|)} \]

- Store these scales in a texture.
- Ideally use to adjust gloss map.
Cost of adding an extra texture too high.

- Gloss maps not used in every shader.
  - Could not combine Toksvig map with existing gloss map.

- Free channels in our DXT5 compressed normal maps:
  
  | 0xFF | Y | 0x00 | X |

Cost of an extra map was too high for both performance and memory.
In Photoshop, the artists could paint a gloss map and place it into the alpha channel of their normal map. On export, Toksvig was applied and the result was placed into the red channel of the DXT5 compressed normal map.
Toksvig Maps

- Compression of normal y component affected:

  ![Without gloss](image1.png)  ![With gloss](image2.png)
Toksvig Maps

- Compression of normal y component affected:

Without gloss

With gloss
Averaging Toksvig to a single value allowed us to have some form of specular filtering with no extra textures and no compression artefacts.
This is just using the average Toksvig factor.
Conclusions

- Calibrate your materials!
- Physically-based is a great starting point:
  - Tweaking is needed.
  - Not everything is simulated.
  - Good tools are essential.
- Can make the change late in a project.
- Possible on current-generation consoles.
References

- [Malin11], Paul Malin, personal communication
Thanks

- Naty Hoffman, 2K Games (@renderwonk)
- Paul Malin, Activision Central Tech (@p_malin)
- Stephen Hill, Ubisoft Montréal (@self_shadow)
- Philippe Gagnon, Ubisoft Montréal
- Mickael Gilabert, Ubisoft Montréal (@mickaeligilabert)
- Vincent Jean, Ubisoft Montréal
- Minjie Wu, Ubisoft Montréal
- Derek Nowrouzezahrai, Université de Montréal
Any Questions?

stephen.mcauley@ubisoft.com
@steveMcauley
http://blog.steveMcauley.com/