Optimizing PBR

Renaldas Zioma
Unity Technologies
Talk Overview

• PBR challenges on Mobile

• What hardware are we optimizing for?

• Faster BRDF

• Linear/Gamma

• Environment Reflections
PBR challenges on Mobile

- Performance
- Many GPUs, many architectures, many peculiarities
- Gamma/Linear workflows
- Lack of high quality texture compression formats
  - ASTC - light at the end of the tunnel
PBR challenges on Mobile

• Shader compilers are still not as good as on PC
• Scalar (more recent) vs vector pipeline
• texCUBEloD
• FP32 vs FP16 precision
• Lots of shader variations!
Optimization Target

based on # of apps running Unity
### Performance

<table>
<thead>
<tr>
<th>Performance Range</th>
<th>PowerVR</th>
<th>NVIDIA</th>
<th>Qualcomm</th>
<th>ARM</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 ~ 8 GFlops</td>
<td>SGX535</td>
<td>Tegra2</td>
<td>Adreno2xx</td>
<td>Mali400 MPx</td>
</tr>
<tr>
<td>0.2 ~ 1 GP/s</td>
<td>iPad, iPhone4</td>
<td></td>
<td></td>
<td>SGS3 (I9300) SGS2 (I9100)</td>
</tr>
<tr>
<td>16 GFlops</td>
<td>SGX54x</td>
<td>Tegra3</td>
<td>Adreno305</td>
<td>MaliT628</td>
</tr>
<tr>
<td>2 ~ 3 GP/s</td>
<td>iPad2/3, iPhone4s, iPhone5</td>
<td></td>
<td>SGS4 mini (I9195)</td>
<td></td>
</tr>
<tr>
<td>100 GFlops</td>
<td>G6x30</td>
<td>Tegra4</td>
<td>Adreno3x0</td>
<td>MaliT760</td>
</tr>
<tr>
<td>4 GP/s</td>
<td>iPadAir, iPhone5s</td>
<td></td>
<td>Nexus 4, Nexus 5</td>
<td>SGS6</td>
</tr>
<tr>
<td>250 GFlops</td>
<td>G6x50</td>
<td>K1, X1</td>
<td>Adreno420</td>
<td></td>
</tr>
<tr>
<td>4 ~ 8 GFlops</td>
<td>iPadAir2, iPhone6</td>
<td></td>
<td>Nexus 9, Shield Tablet</td>
<td></td>
</tr>
</tbody>
</table>

- Huge performance leap in every generation
## Market Share

<table>
<thead>
<tr>
<th>PowerVR</th>
<th>NVIDIA</th>
<th>Qualcomm</th>
<th>ARM</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 ~ 8 GFlops 0.2 ~ 1 GP/s</td>
<td>SGX535</td>
<td>Tegra2</td>
<td>Adreno2xx</td>
</tr>
<tr>
<td>16 GFlops 2 ~ 3 GP/s</td>
<td>SGX54x</td>
<td>Tegra3</td>
<td>Adreno305</td>
</tr>
<tr>
<td>100 GFlops 4 GP/s</td>
<td>G6x30</td>
<td>Tegra4</td>
<td>Adreno3x0</td>
</tr>
<tr>
<td>250 GFlops 4 ~ 8 GP/s</td>
<td>G6x50</td>
<td>K1, X1</td>
<td>Adreno420</td>
</tr>
</tbody>
</table>

- **Green** - GPU with significant market share
- **TIP**: new devices >10x faster than what most people have in their pocket!
### Optimization Tiers

<table>
<thead>
<tr>
<th></th>
<th>PowerVR</th>
<th>NVIDIA</th>
<th>Qualcomm</th>
<th>ARM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Low-end Tier</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SGX535</td>
<td>3.5%</td>
<td>Tegra2</td>
<td>Adreno2xx</td>
<td>9%</td>
</tr>
<tr>
<td><strong>Mid Tier</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SGX54x</td>
<td>15.4%</td>
<td>Tegra3</td>
<td>Adreno305</td>
<td>7.1%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Mali400 MPx 19%</td>
</tr>
<tr>
<td><strong>High-end Tier</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G6x30</td>
<td>6.0%</td>
<td>Tegra4</td>
<td>Adreno3x0</td>
<td>10.3%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MaliT628 0.5%</td>
</tr>
<tr>
<td>G6x50</td>
<td>0.3%</td>
<td>K1, X1</td>
<td>Adreno420</td>
<td>0.1%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MaliT760 0.0%</td>
</tr>
</tbody>
</table>

- • iOS, Android and Windows combined
Important GPU characteristics for PBR

- Ratio between math (ALU) and fetching texture (TEX)
- Scalar or vector architecture
- Precision
<table>
<thead>
<tr>
<th>PowerVR</th>
<th>NVIDIA</th>
<th>Qualcomm</th>
<th>ARM</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 GFlops</td>
<td><strong>SGX54x</strong> 16 FLOPs / 1 TEX FP16 * vector</td>
<td></td>
<td><strong>Mali400 MPx</strong> 16 FLOPs / 1 TEX FP16 only vector</td>
</tr>
<tr>
<td>70 GFlops</td>
<td><strong>SGX554</strong> 32 FLOPs / 1 TEX FP16 * vector</td>
<td><strong>Adreno3xx</strong> 16? FLOPs / 1 TEX FP32/FP16 scalar</td>
<td><strong>MaliT604</strong> 16* FLOPs / 1 TEX FP32-FP16 wide vector</td>
</tr>
<tr>
<td>&gt;100 GFlops</td>
<td><strong>G6x30</strong> 48 FLOPs / 1 TEX FP32-FP16 scalar</td>
<td><strong>K1</strong> 48 FLOPs / 1 TEX FP32 only scalar</td>
<td><strong>MaliT628</strong> 32* FLOPs / 1 TEX FP32-FP16 wide vector</td>
</tr>
<tr>
<td>&gt;200 GFlops</td>
<td><strong>G6x50</strong> 64 FLOPs / 1 TEX FP32-FP16 scalar</td>
<td><strong>X1</strong> 64/128 FLOPs / 1 TEX FP32-FP16 scalar</td>
<td><strong>MaliT760</strong> &gt;68 FLOPs / 1 TEX FP32-FP16 wide vector</td>
</tr>
</tbody>
</table>

- Unofficial numbers, some based on our measurements. Numbers might be wrong! Numbers are peak values.
- **TEX** - bilinear texture fetch
- **FP32-FP16** - supports both precision, likely to be faster in FP16
- **FP16** * - definitely faster in FP16, but certain complex operations (EXP, LOG, etc) will be executed in FP32 anyway
- **wide vector** - FP16 are likely to be executed as 8-way vectors
Important GPU characteristics for PBR

- FP16 ("FP16 only" & "FP16 *")
  - PBS is more prone to artifacts @ low precision
  - Check your epsilons (1e-4 is OK, 1e-5 is not!)
  - Sometimes need additional clamping due to precision overflows
  - Vector pipeline might need different optimizations
  - ALU/TEX differs a lot for high-end vs low-end GPUs
## Optimizing for High-end tier

<table>
<thead>
<tr>
<th>High-end Tier</th>
<th>PowerVR</th>
<th>NVIDIA</th>
<th>Qualcomm</th>
<th>ARM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGX535</td>
<td>3.5%</td>
<td>Tegra2</td>
<td>Adreno2xx</td>
<td>9%</td>
</tr>
<tr>
<td>SGX54x</td>
<td>15.4%</td>
<td>Tegra3</td>
<td>Adreno305</td>
<td>7.1%</td>
</tr>
<tr>
<td>G6x30</td>
<td>6.0%</td>
<td>Tegra4</td>
<td>Adreno3x0</td>
<td>10.3%</td>
</tr>
<tr>
<td>G6x50</td>
<td>0.3%</td>
<td>K1, X1</td>
<td>Adreno420</td>
<td>0.1%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MaliT628</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MaliT760</td>
</tr>
</tbody>
</table>
Optimizing BRDF for Mobile

\[ I_{spec} = \frac{D(N \cdot H, roughness) \cdot G(N \cdot V, N \cdot L, roughness) \cdot F(L \cdot H, specColor)}{4 \cdot (N \cdot V) \cdot (N \cdot L)} \cdot N \cdot L \]

- Specular micro-facet equation
- D: Distribution Term
- **GGX vs Normalized Blinn-Phong vs SG approx.**
- V: Visibility term
- F: Fresnel term

\[ V = \frac{G(N \cdot V, N \cdot L, roughness)}{4 \cdot (N \cdot V) \cdot (N \cdot L)} \]
GGX vs BlinnPhong

• GGX - more simple ops (ADD, MUL), but only 1 complex (RCP)

\[
GGX = \frac{\text{roughness}^4}{\pi \left( (N \cdot H)^2 \left( \text{roughness}^4 - 1 \right) + 1 \right)^2}
\]

• Normalized Phong - several complex ops (RCP, EXP, LOG)

\[
Phong = \frac{1}{\pi \text{roughness}^4 \cdot (N \cdot H) \left( \frac{2}{\text{roughness}^4} - 2 \right)}
\]

• even SG approximation (RCP, EXP)
Simple vs Complex op

- PowerVR G6x00 asm (Phong example)
- Can do many ops / cycle, but only 1 complex!
- Most other architectures complex op = latency

```
23   : fmad ft0, i0, r22, r9
    fmul ft1, c71, r13
    pck.f32 ft2
    tstgz.f32 ftt, _, ft0
    mov i0.e0.e1.e2.e3, i3, ftt, ft0, ft1

24   : flog i0, i0.abs

25   : fmul ft0, i1, i2
    fmul ft1, i0, i3
    mov i3, ft0;
    mov i0, ft1;

26   : fexp i0, i0

27   : fadd ft0, i3, r23
    fmul ft1, i0, r23
    mov i2, ft0;
    mov i1, ft1;
```
Geometric / Visibility term

- Smith adopted for GGX
  \[ V_{\text{Smith}} = \frac{1}{((N \cdot L)(1-k)+k)((N \cdot V)(1-k)+k)} \]

- Kelemen and Szirmay-Kalos (KSK)
  \[ V_{\text{SKS}} = \frac{1}{(L \cdot H)(L \cdot H)} \]
  does not take roughness into account!

- Fix for KSK (J. Hable)
  \[ V_{\text{SKS}^m} = \frac{1}{(L \cdot H)^2(1 - \text{roughness}^2) + \text{roughness}^2} \]
  dependent only on \( \mathbf{L} \cdot \mathbf{H} \) and Roughness!
Fresnel term

• Approximation suggested by C. Schüler:
  \[ F = \frac{specColor}{L \cdot H} \]

• Dielectrics - OK
  (reflectance 0.02 ~ 0.15)

• Conductors aka Metals - average value OK
  (reflectance 0.7 ~ 1.0)

  • has wrong shape, but Fresnel is almost flat for Metals anyway

• Goes to +Infinity instead of 1
Fresnel term

\[
F = \frac{\text{specColor}}{L \cdot H}
\]

- Will not use Schüler approximation directly
- Just inspiration that specColor can be post multiplied
  - Great for scalar pipeline!
V*F together

- Modified KSK and Schlick Fresnel depend on $\mathbf{L}\cdot\mathbf{H}$
- Fuse them together

\[
V \cdot F = \frac{(1-L \cdot H)^5}{(L \cdot H)^2(1-\text{roughness}^2)+\text{roughness}^2}
\]
- Cheaper approximation?
Approximate $V\cdot F$

- Not an algebraic simplification
- Fitting similar curve

$$V\cdot F_{\text{approx}} = \frac{1}{(L\cdot H)^2 \cdot (\text{roughness} + 0.5)} \cdot \text{specColor}$$
Approximation Results

Original (Modified KSK, Fresnel)

Our approximation

Errors
Approximate $V^*F$

- Good for Dielectrics, but diverge for Metals

Original (Modified KSK, Fresnel)  
Our approximation
Approximate V*F

- Can be improved with couple more ops, but does not matter in practice

Original (Modified KSK, Fresnel)  

Our approximation*
Comparison of Visibility Terms

<table>
<thead>
<tr>
<th></th>
<th>Implicit + Fresnel</th>
<th>Smith + Fresnel</th>
<th>Our Approximation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete lighting</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Plastic</td>
<td>⏳</td>
<td>⏳</td>
<td>Plastic</td>
</tr>
<tr>
<td>Metal</td>
<td>⏳</td>
<td>⏳</td>
<td>Metal</td>
</tr>
<tr>
<td>V*F terms only</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Plastic</td>
<td>⏳</td>
<td>⏳</td>
<td>Plastic</td>
</tr>
<tr>
<td>Metal</td>
<td>⏳</td>
<td>⏳</td>
<td>Metal</td>
</tr>
</tbody>
</table>
Final Specular BRDF

\[
BRDF_{\text{spec}} = \frac{\text{roughness}^4}{4 \cdot \pi \left( (N \cdot H)^2 (\text{roughness}^4 - 1) + 1 \right)^2 \cdot (L \cdot H)^2 (\text{roughness} + 0.5)} \cdot \text{specColor}
\]

- Just 1 division
- Good for scalar pipeline
Environment BRDF

- B. Karis approximation based on D. Lazarov work
- Just refitted with simpler function

\[
BRDF_{env} = (\max(\text{roughness}, N \cdot V))^3 + \text{specColor}
\]
Environment BRDF

\[(1 - \text{max}(\text{roughness}, N \cdot V))^3\]
Putting everything together
Putting everything together

<table>
<thead>
<tr>
<th></th>
<th>ImgTech G6x00 (scalar)</th>
<th>ImgTech SQX554 (vector)</th>
<th>QCOM Adreno305 (scalar)</th>
<th>ARM Mali T760 (vector)</th>
<th>ARM Mali400MP4 (vector)</th>
</tr>
</thead>
<tbody>
<tr>
<td>old-school-non-PBR</td>
<td>141%</td>
<td>172%</td>
<td>154%</td>
<td>140%</td>
<td></td>
</tr>
<tr>
<td>normalized BlinnPhong, Smith (baseline)</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>proposed version GGX</td>
<td>114%</td>
<td>126%</td>
<td>118%</td>
<td>111%</td>
<td>271%</td>
</tr>
</tbody>
</table>

- Percentages are used to make test runs on different screen resolutions easily comparable.
- Measured with a scene consisting of 50K vertices fully covering screen with >3x overdraw rate.
## Optimizing for Mid tier

<table>
<thead>
<tr>
<th>PowerVR</th>
<th>NVIDIA</th>
<th>Qualcomm</th>
<th>ARM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGX535</td>
<td>Tegra2</td>
<td>Adreno2xx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3.5%</td>
<td>1.0%</td>
<td></td>
</tr>
<tr>
<td>SGX54x</td>
<td>Tegra3</td>
<td>Adreno305</td>
<td>Mali400 MPx</td>
</tr>
<tr>
<td></td>
<td>15.4%</td>
<td>0.9%</td>
<td>19%</td>
</tr>
<tr>
<td>G6x30</td>
<td>Tegra4</td>
<td>Adreno3x0</td>
<td>MaliT628</td>
</tr>
<tr>
<td></td>
<td>6.0%</td>
<td>0.0%</td>
<td>0.5%</td>
</tr>
<tr>
<td>G6x50</td>
<td>K1, X1</td>
<td>Adreno420</td>
<td>MaliT760</td>
</tr>
<tr>
<td></td>
<td>0.3%</td>
<td>0.0%</td>
<td>0.0%</td>
</tr>
</tbody>
</table>
Per-vertex lighting

• Medium-end hardware:
  • Lower bandwidth, GFLOPs are meh
• Diffuse and ambient per-vertex
• Specular per-pixel
• Environment reflection vector per-vertex
• Specular in Tangent space - saves matrix-vector transformation
### Optimizing for Low-end Tier

<table>
<thead>
<tr>
<th>PowerVR</th>
<th>NVIDIA</th>
<th>Qualcomm</th>
<th>ARM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGX535</td>
<td>Tegra2</td>
<td>Adreno2xx</td>
<td>Mali400 MPx</td>
</tr>
<tr>
<td>3.5%</td>
<td>1.0%</td>
<td>9%</td>
<td>19%</td>
</tr>
<tr>
<td>SGX54x</td>
<td>Tegra3</td>
<td>Adreno305</td>
<td>MaliT628</td>
</tr>
<tr>
<td>15.4%</td>
<td>0.9%</td>
<td>7.1%</td>
<td>0.5%</td>
</tr>
<tr>
<td>G6x30</td>
<td>Tegra4</td>
<td>Adreno3x0</td>
<td>MaliT760</td>
</tr>
<tr>
<td>6.0%</td>
<td>0.0%</td>
<td>10.3%</td>
<td>0.0%</td>
</tr>
<tr>
<td>G6x50</td>
<td>K1, X1</td>
<td>Adreno420</td>
<td></td>
</tr>
<tr>
<td>0.3%</td>
<td>0.0%</td>
<td>0.1%</td>
<td></td>
</tr>
</tbody>
</table>
LUT

• Low-end hardware:
  • Low ALU/TEX ratio

• Specular intensity in LUT
  • $\langle \mathbf{N} \cdot \mathbf{H}, \text{Roughness} \rangle$

• Remember implicit Geometric term!
  • $I = \text{BRDF} \times \mathbf{N} \cdot \mathbf{L}$

• $\mathbf{N} \cdot \mathbf{H}$ is cosine - highlights are really crammed
LUT specular

• Store 1/16 intensity in LUT

• $\textbf{R} \cdot \textbf{L}$ instead on $\textbf{N} \cdot \textbf{H}$ saves couple of ops
  • suggested by B.Karis

• Warp LUT /w $\textbf{R} \cdot \textbf{L}^4$ to get more space for highlights
• PBR challenges on Mobile
• What hardware are we optimizing for?
• Faster BRDF

• Linear/Gamma
• Environment Reflections
Linear/Gamma

- Linear lighting
  - hard on older GPUs
  - has additional cost
- Gamma and Linear will never look the same, but we can aim for:
  - consistent base light intensity
  - consistent highlight size
Hack for Gamma to “match” Linear

- Approximate gamma with 2.0
- “Fixup” just **specular intensity**:
  - Keep parameters (Roughness) for specular part of equation in Linear
  - Evaluate specular intensity as in Linear space
  - Convert resulting specular intensity to sRGB space before applying colors:
    \[
    = \sqrt{\text{specIntensity}_\text{Linear}} \times \text{specColor}_\text{sRGB}
    \]
Pros of Gamma hack

• No need to uncompress colors/textures from sRGB to Linear

• Roughness is Linear already
  • usually stored in Alpha channel

• Potentially long latency op (INVSQRT) is NOT at the end of the shader
  • cost can be hidden by other ops
Gamma vs Linear
Environment reflections

• texCUBElod can be really expensive sometimes
  • G6xx0 - high-end mobile GPU!
  • optional extension on ES2.0
  • G6xx0: use dynamic branches to pick 2 closest mips and lerp
    • slightly faster!
texCUBElod

- Lerp 2 extreme mips
  - ugly, but fast

- 3-way lerp:
  - hardcoded highest mip#
  - middle mip#
  - 2nd order SH

- for middle you can cut mip levels (/w extension) and hardcode to a very large number
Thanks

John Hable
Morten Mikkelsen
Florian Penzkofer
Alexey Orlov
Dominykas Kiauleikis
Sakari Pitkänen
References


4. Brian Karis, “Physically Based Shading on Mobile”, 2014, online


7. Robert Cook and Kenneth Torrance, “A reflectance model for computer graphics”
Bonus Slides
## OpenGL ES3.0

<table>
<thead>
<tr>
<th>PowerVR</th>
<th>NVIDIA</th>
<th>Qualcomm</th>
<th>ARM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>4 ~ 8 GFlops</strong>&lt;br&gt;0.2 ~ 1 GP/s</td>
<td>SGX535</td>
<td>Tegra2</td>
<td>Adreno2xx</td>
</tr>
<tr>
<td></td>
<td>iPad, iPhone4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>16 GFlops</strong>&lt;br&gt;2 ~ 3 GP/s</td>
<td>SGX54x</td>
<td>Tegra3</td>
<td>Adreno305</td>
</tr>
<tr>
<td></td>
<td>iPad2/3, iPhone4s, iPhone5</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>100 GFlops</strong>&lt;br&gt;4 GP/s</td>
<td>G6x30</td>
<td>Tegra4</td>
<td>Adreno3x0</td>
</tr>
<tr>
<td></td>
<td>iPadAir, iPhone5s</td>
<td></td>
<td>Nexus 4, Nexus 5</td>
</tr>
<tr>
<td><strong>250 GFlops</strong>&lt;br&gt;4 ~ 8 GP/s</td>
<td>G6x50</td>
<td>K1, X1</td>
<td>Adreno420</td>
</tr>
<tr>
<td></td>
<td>iPadAir2, iPhone6</td>
<td>Nexus 9, Shield Tablet</td>
<td></td>
</tr>
</tbody>
</table>

- Green - GPU with ES3.0 support
- TIP: you can’t just use ES2.0 / ES3.0 to determine performance of GPU
### Low-end with large share

<table>
<thead>
<tr>
<th>PowerVR</th>
<th>NVIDIA</th>
<th>Qualcomm</th>
<th>ARM</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 ~ 8 GFlops 0.2 ~ 1 GP/s</td>
<td>SGX535 3.5%</td>
<td>Tegra2 1.0%</td>
<td>Adreno2xx 9%</td>
</tr>
<tr>
<td>16 GFlops 2 ~ 3 GP/s</td>
<td>SGX54x 15.4%</td>
<td>Tegra3 0.9%</td>
<td>Adreno305 7.1%</td>
</tr>
<tr>
<td>100 GFlops 4 GP/s</td>
<td>G6x30 6.0%</td>
<td>Tegra4 0.0%</td>
<td>Adreno3x0 10.3%</td>
</tr>
<tr>
<td>250 GFlops 4 ~ 8 GP/s</td>
<td>G6x50 0.3%</td>
<td>K1, X1 0.0%</td>
<td>Adreno420 0.1%</td>
</tr>
</tbody>
</table>

- Yellow - Low-end with large share, but most in APAC and Latin America
- And you still need to support iPhone4
Textures

• Lack of uncorrelated 4 channel compression

• Consider Roughness in a separate texture

• Pairing Roughness with Specular/Metal instead of Albedo or Normals since former is low frequency & low variance data
Textures

- Lack of HDR compression
- IBL, Lightmaps: RGB*2 instead of RGBm/HDR
- IBL: uncompressed HDR cubemaps
- An awful tradeoff :(