ASTC
(Adaptive Scalable Texture Compression)

Daniele Di Donato
Senior Software Engineer
Outline

- Introduction to texture compression and block-based algorithms
- Previous texture compression formats vs. ASTC
- How to tune quality and size in ASTC
- Use cases for ASTC 2D and 3D textures
Texture Compression

- Modern graphic applications use lots of textures

Reflectance

Normals

Gloss, Height, etc

Illuminance

Lighting environment

3D Properties
Texture Compression

- …but textures are big
  - At 24 bits/pixel, multi-megabytes

- This is a problem
  - Big downloads
  - Big memory footprint
  - Big memory bandwidth
  - Big power drain

- So we have to compress them
Block-Based Compression

- Texture compression methods all work basically the same way
  - Split image into blocks
    - Random access
    - Fast
  - Decode pixel from block
    - Fixed data access cost
    - Fixed bit rate
  - Blocks have two sizes
    - Footprint (NxM pixels)
    - Data size (n bits)
  - Lossy!
Complete Guide to Texture Compression (Abridged)

- Texture data is big

- 32bpp * 1024 * 1024 = 4MB per texture

- This is the reason for jpg and png

- Hardware decompression saves memory and bandwidth
Complete Guide to Texture Compression (Abridged)

- Hardware needs random access
- Texture compression is block based
- Look up a block from the texel coords
- Decompress into local cache
- Sample cached block
Complete Guide to Texture Compression (Abridged)

- Details about block must be known
- Block footprint dimensions
- Block size in memory
- How the block is encoded
- There are many, *many* encodings
Different codecs support different footprints and bitrates

They also support different color encodings (HDR / sRGB / alpha)

Hardware limits access to proprietary codecs
What Makes ASTC Special

- Wide range of bitrates
- Wide range of formats
- Handles sRGB
- Handles HDR
- 3D Textures
- Non Proprietary
What Defines Quality?

- Compression process is asymmetric
  - Tile extracted from data points
  - Finding right points is mostly trial and error

- Quality decided by 3 factors
  - Precision of data points (bit rate)
  - Number of attempts per tile (limits)
  - Types of error to reject/ignore (priority)
What Defines Quality?

Bit rates and block size

- Every block in ASTC has the same data size
  - 128 bits

- Bit rate is decided by block footprint.
  - From 4x4 (8 bpp) to 12x12 (0.89 bpp)

- More data per pixel, closer to original

- Saves less bandwidth
What Defines Quality?

Limits and leeway

- Trial and error process
  - More tries, more opportunities to find a good match
  - Takes longer to compress

- Error bounds and quality limits
  - Move on if the early tries look bad
  - Stick with an early try if it is ‘good enough’
  - Stop looking if a block takes too long

- Mostly managed through presets
What Defines Quality?

Priority and perception

- Weightings on specific error types
What Defines Quality?

Priority and perception

- Weightings on specific error types
- Per channel weighting
What Defines Quality?

Priority and perception

- Weightings on specific error types
- Per channel weighting
- Block weighting
What Defines Quality?

Priority and perception

- Weightings on specific error types
- Per channel weighting
- Block weighting
- Angular weighting
What Defines Quality?

Priority and perception

- Weightings on specific error types
- Per channel weighting
- Block weighting
- Angular weighting
- Absolute value weighting

\[ error = \frac{1}{\text{baseweight} + \text{avgscale} \times \text{average}^2 + \text{stdevscale} \times \text{stdev}^2} \]
2D Texture Support
Available block sizes for 2D texture

- Various block sizes available
- More flexible than previous algorithm
- Allows texture quality LOD
2D Texture Support
Use case: porting SeeMore demo to ASTC

Compared to ETC2:

- Using ASTC with 5x5 block size gave us ~24% memory reduction maintaining the same quality level (measured using PSNR).

- Using ASTC 6x6 gave us a ~45% memory reduction with negligible quality loss.

- As a consequence, the memory read bandwidth required for texturing reduced, improving the performance.

<table>
<thead>
<tr>
<th>Textures</th>
<th>Total Size MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uncompressed</td>
<td>263.00</td>
</tr>
<tr>
<td>ETC2</td>
<td>93.10</td>
</tr>
<tr>
<td>ASTC 5x5</td>
<td>71.00</td>
</tr>
<tr>
<td>ASTC 6x6</td>
<td>49.40</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Memory Read Bandwidth in MB/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uncompressed</td>
</tr>
<tr>
<td>ETC2</td>
</tr>
<tr>
<td>ASTC 5x5</td>
</tr>
<tr>
<td>ASTC 6x6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Energy consumption DDR2 mJ per frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uncompressed</td>
</tr>
<tr>
<td>ETC2</td>
</tr>
<tr>
<td>ASTC 5x5</td>
</tr>
<tr>
<td>ASTC 6x6</td>
</tr>
</tbody>
</table>
3D Texture Support

Use case: ASTC 3D texture for particles collision

- Various block sizes to choose from
- Each compressed block will still occupy 128 bits
- HDR support allows us to store 16Bit half-floats

<table>
<thead>
<tr>
<th>Block Dimension</th>
<th>Bit Rate (bits per texel)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3x3x3</td>
<td>4.74</td>
</tr>
<tr>
<td>4x3x3</td>
<td>3.56</td>
</tr>
<tr>
<td>4x4x3</td>
<td>2.67</td>
</tr>
<tr>
<td>4x4x4</td>
<td>2.00</td>
</tr>
<tr>
<td>5x4x4</td>
<td>1.60</td>
</tr>
<tr>
<td>5x5x4</td>
<td>1.28</td>
</tr>
<tr>
<td>5x5x5</td>
<td>1.02</td>
</tr>
<tr>
<td>6x5x5</td>
<td>0.85</td>
</tr>
<tr>
<td>6x6x5</td>
<td>0.71</td>
</tr>
<tr>
<td>6x6x6</td>
<td>0.59</td>
</tr>
</tbody>
</table>
3D Texture Support

Use case: ASTC 3D texture for particles collision

- Since the simulation is run in the vertex shader we need to provide information about the environment to collide with.

- Voxelizing the environment allows us to save it as 3D texture.

- But the data generated is huge….so let’s compress it with ASTC 3D.
3D Texture Support

Use case: ASTC 3D texture for particles collision

- ~90% memory reduction
- ~38% memory bandwidth reduction
Why Not Try ASTC Right Now?

- Command line compressor
  - ASTC Evaluation Codec

- GUI compressor
  - Mali Texture Compression Tool

- Lacking compatible hardware?
  - Mali OpenGL® ES 3.1 Emulator

Mali Developer Center:
MaliDeveloper.arm.com
To Find Out More….

- Visit the demo area behind this lecture theatre
  - Find the Ice Cave demo
  - In-depth Q&A with ARM engineers

  - Revisit this talk post GDC
  - Download tools and resources
More Talks from ARM at GDC 2015

Available post-show online at Mali Developer Center

- **Unreal Engine 4 mobile graphics and the latest ARM CPU and GPU architecture** - Weds 9:30AM; West Hall 3003
  This talk introduces the latest advances in features and benefits of the ARMv8-A and tile-based Mali GPU architectures on Unreal Engine 4, allowing mobile game developers to move to 64-bit’s improved instruction set.

- **Unleash the benefits of OpenGL ES 3.1 and Android Extension Pack (AEP)** – Weds 2PM; West Hall 3003
  OpenGL ES 3.1 provides a rich set of tools for creating stunning images. This talk will cover best practices for using advanced features of OpenGL ES 3.1 on ARM Mali GPUs using recently developed examples from the Mali SDK.

- **Making dreams come true – global illumination made easy** – Thurs 10AM; West Hall 3014
  In this talk, we present an overview of the Enlighten feature set and show through workflow examples and gameplay demonstrations how it enables fast iteration and high visual quality on all gaming platforms.

- **How to optimize your mobile game with ARM Tools and practical examples** – Thurs 11:30AM; West Hall 3014
  This talk introduces you to the tools and skills needed to profile and debug your application by showing you optimization examples from popular game titles.

- **Enhancing your Unity mobile game** – Thurs 4PM; West Hall 3014
  Learn how to get the most out of Unity when developing under the unique challenges of mobile platforms.

More Talks from ARM at GDC 2015
Thank you
Any questions?