On improving the performance per area of ASTC with a multi-output decoder

Kenneth Rovers

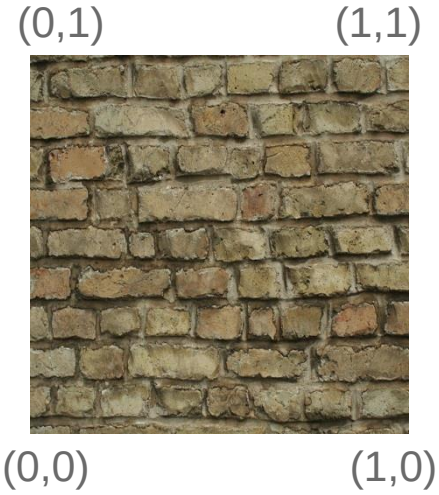
25th July, 2017

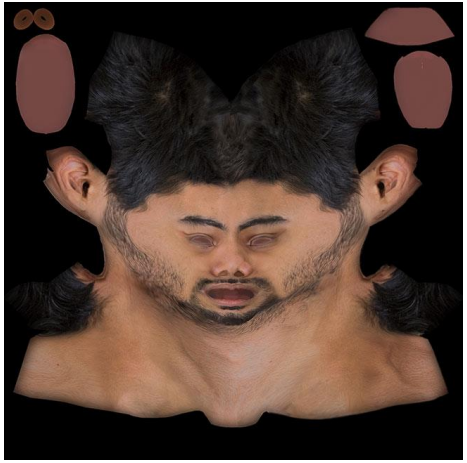
www.imgtec.com

Textures

Background

- **In general, an image to be mapped to an object surface**
  - But also lighting, normals, height, etc.
- **Texels are texture pixels**
  - Pixels are picture elements

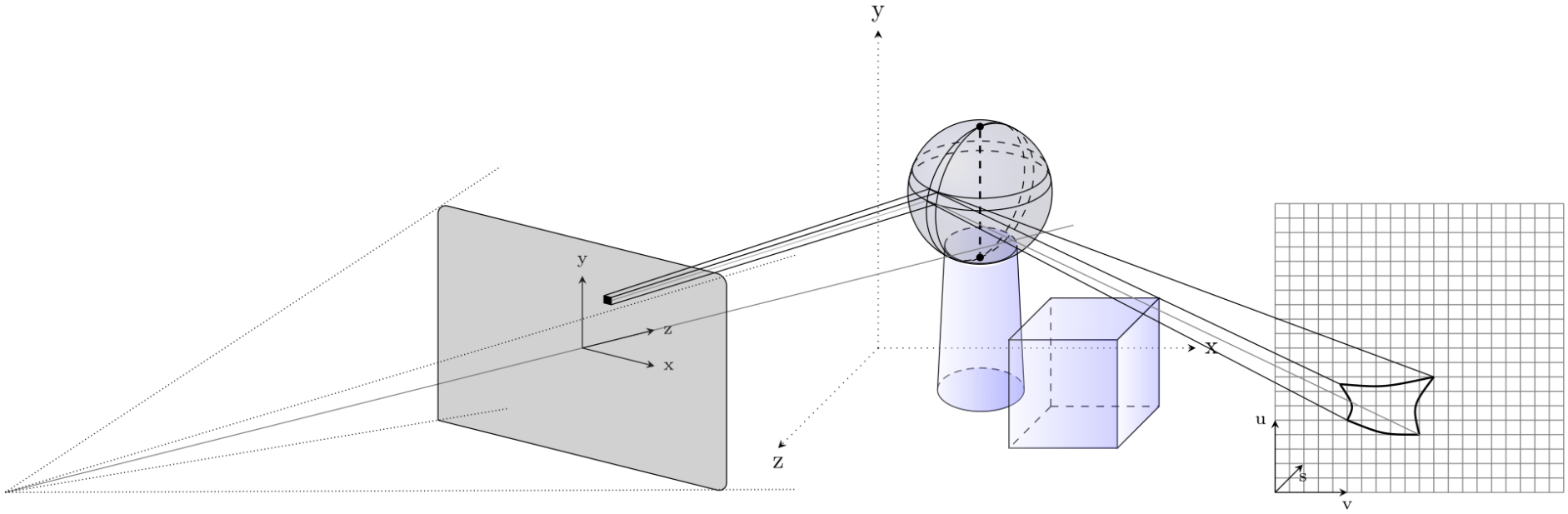




Textures

Background

- Texels do not exactly map to (screen) pixels
- Each screen pixel (fragment) may require up to 128 texels



Texture compression

Motivation

- GPUs are in general memory bandwidth limited
- Textures consume a large part of this bandwidth
- Larger resolutions means larger textures means more bandwidth

- Compression of textures is essential

Texture compression

Requirements

- **Good compression**
  - typically means it may be lossy at the cost of quality
- **Good quality**
  - few artefacts
- **Flexible**
  - Number of colour channels
  - Quality
- **Simple decoding**
  - Smaller implementation

Example of Lossy Compression

Original Lena Image (12KB size)

Lena Image, Compressed (85% less information, 1.8KB)

Lena Image, Highly Compressed (96% less information, 0.56KB)

ASTC

- **Adaptive Scalable Texture Compression**
  - One texture compression format to rule them all
  - Flexible, per-texture, space versus quality trade-off

ASTC

- **Adaptive Scalable Texture Compression**
  - One texture compression format to rule them all
  - Flexible, per-texture, space versus quality trade-off

- **Allows from 4×4 up to 12×12 texels to be compressed in a fixed 128 bits**
- **Allow 1 to 4 colour channels**
  - Modes with different quality per channel possible
- **Improve quality by grouping texels into up to four different partitions**

ASTC

- **Adaptive Scalable Texture Compression**
  - One texture compression format to rule them all
  - Flexible, per-texture, space versus quality trade-off

- **Allows from 4×4 up to 12×12 texels to be compressed in a fixed 128 bits**
- **Allow 1 to 4 colour channels**
  - Modes with different quality per channel possible
- **Improve quality by grouping texels into up to four different partitions**

- **High quality, very flexible, many use cases → no longer simple to decode!**

Approach

- **ASTC decoder is large in area!**

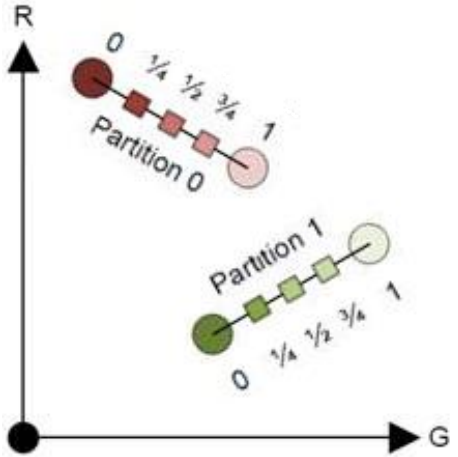
- **(Simple) idea**
  - Nearby texels are likely to be requested at the same time → multi-output decoder
  - Share common decoding hardware → save area

ASTC

Basics

- **Up to four partitions**
- **Each partition has a set of colour endpoints**
- **Each texel has a partition index**
- **Each texel has a weight used to interpolate between the colour endpoints**

- **Decode**
  - Configuration data
  - Weight data
  - Colour data
- **Interpolate**



Texel Weights

0	0	¼	½
0	0	¼	¾
¼	¼	½	1
¾	1	1	1

stored with block

Partition Indices

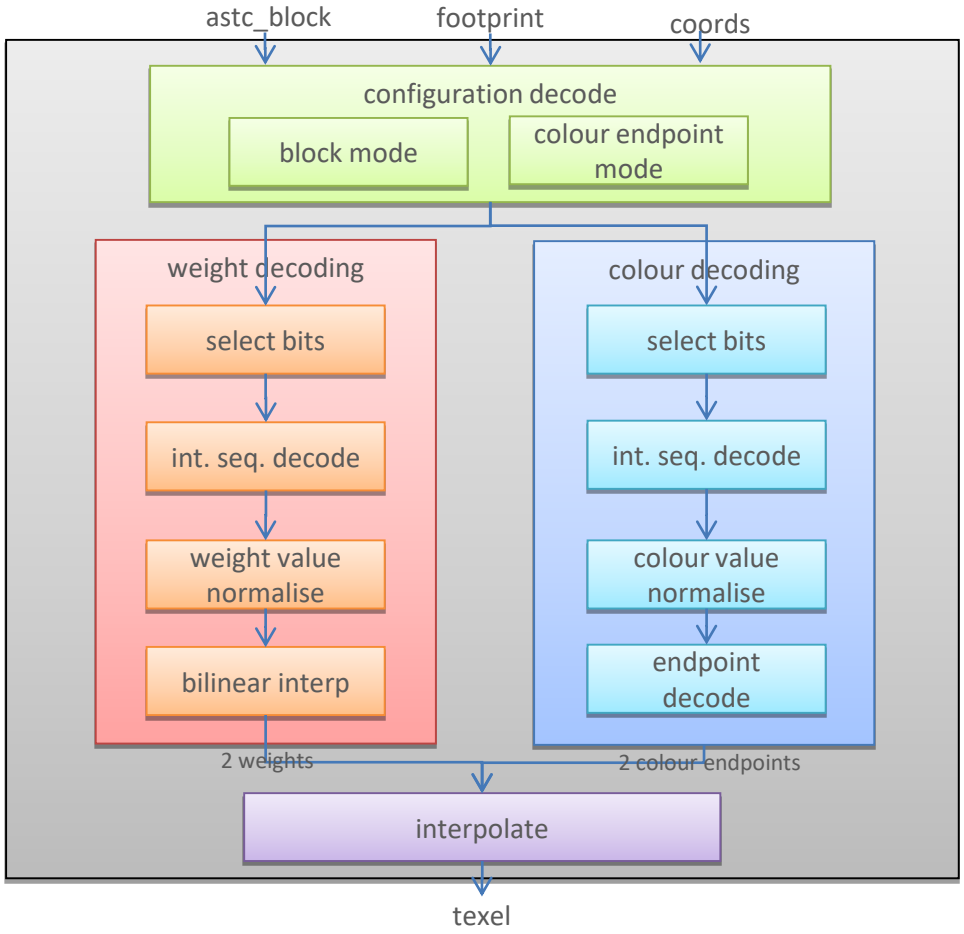
1	0	0	0
1	1	0	0
1	1	1	0
1	1	1	1

pattern index stored with block

ASTC decoder

Structure

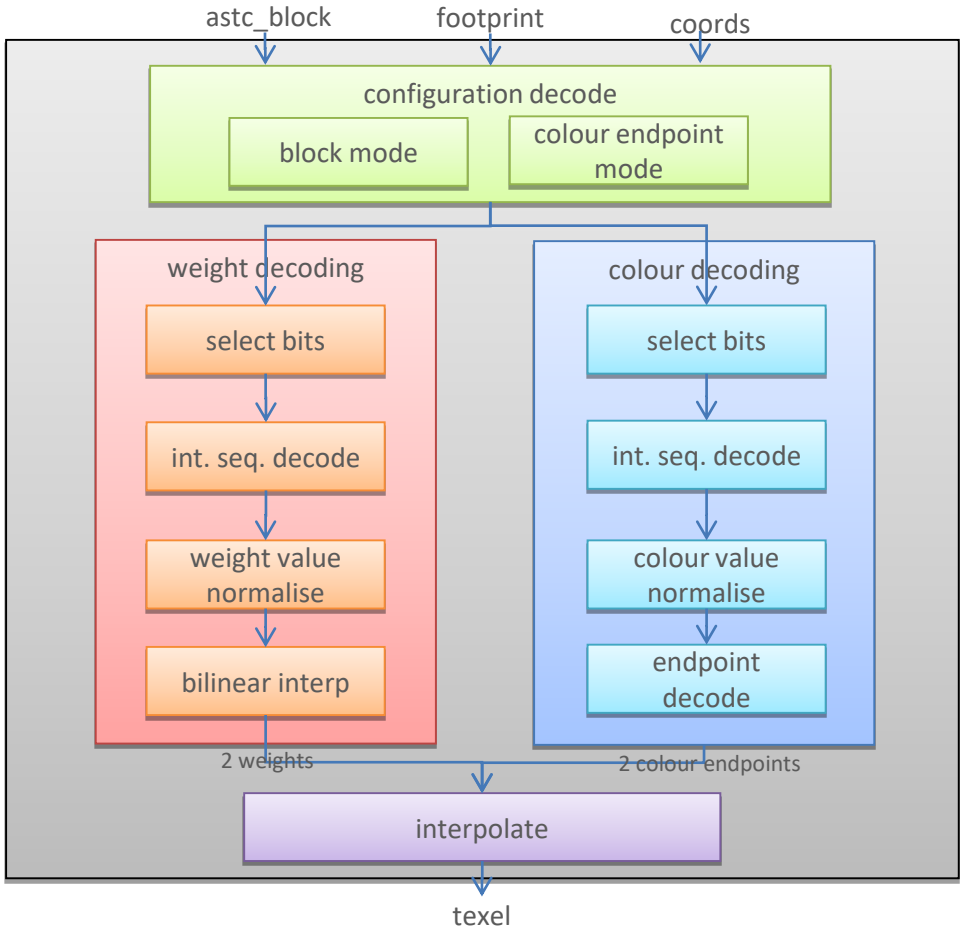
- **Configuration**
  - sizes, modes, partitions
- **Weights**
  - Variable size
  - Fractional number of bits per weight
  - Normalise
  - Fewer weights can be stored than number of texels → bilinear interpolation
  - Results in 1 or 2 weights



ASTC decoder

Structure

- **Colours**
  - Variable size
  - Fractional number of bits per colour value
  - Normalise
  - Colour endpoint mode determines how colour values combine to actual colours
  - Results in two colours
- **Interpolate**
  - Up to four colour channels, each using one of the two weights



ASTC multi-output decoder

2x2 block

- **Assume a four-output decoder**
  - In general any number of outputs possible
- **Assume a 2×2 block of adjacent texels**
  - Very common for texture filtering

Sharing

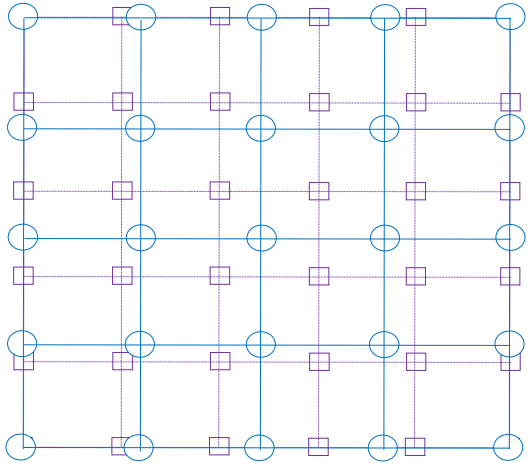
For a 2×2 request

- **Configuration data is mostly shared for all texels**
  - Partition index is per texel
  - Colour endpoint mode is per partition (potentially all needed)

Sharing

For a 2×2 request

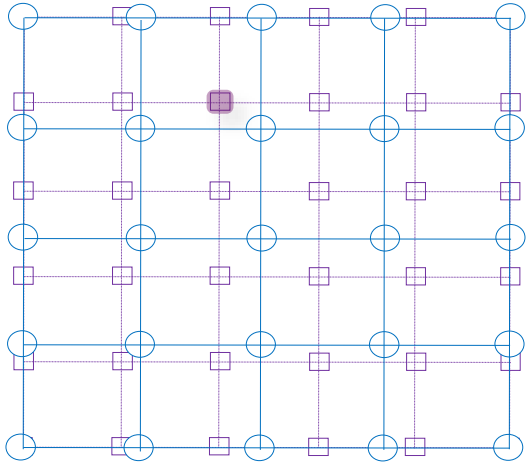
- **Configuration data is mostly shared for all texels**
  - Partition index is per texel
  - Colour endpoint mode is per partition (potentially all needed)
- **Weight data**
  - Bit selection shared
  - Adjacent texels share weight values for bilinear interpolation
    - Instead of 4 texels·4 interpolant·2 weight =32 weight values, only 3·3·2=18 are needed



Sharing

For a 2×2 request

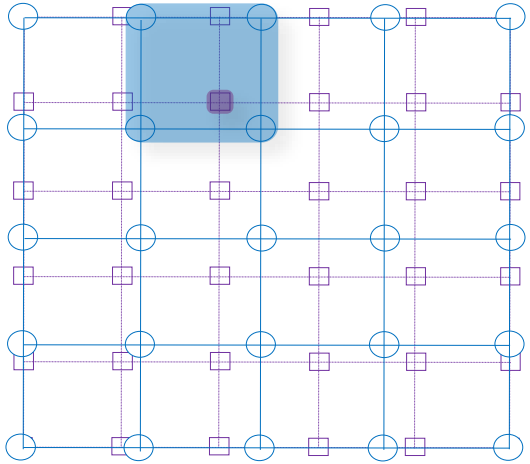
- **Configuration data is mostly shared for all texels**
  - Partition index is per texel
  - Colour endpoint mode is per partition (potentially all needed)
- **Weight data**
  - Bit selection shared
  - Adjacent texels share weight values for bilinear interpolation
    - Instead of 4 texels·4 interpolant·2 weight =32 weight values, only 3·3·2=18 are needed



Sharing

For a 2×2 request

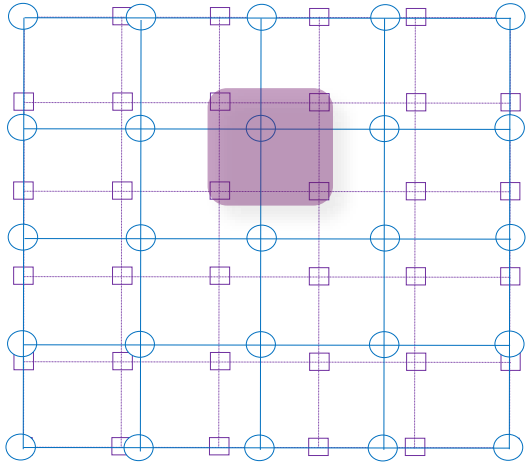
- **Configuration data is mostly shared for all texels**
  - Partition index is per texel
  - Colour endpoint mode is per partition (potentially all needed)
- **Weight data**
  - Bit selection shared
  - Adjacent texels share weight values for bilinear interpolation
    - Instead of 4 texels·4 interpolant·2 weight =32 weight values, only 3·3·2=18 are needed



Sharing

For a 2×2 request

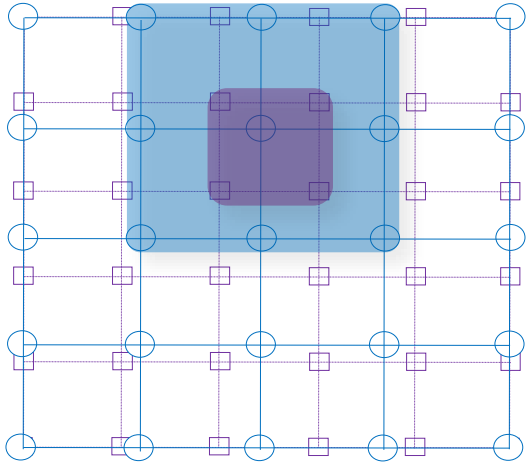
- **Configuration data is mostly shared for all texels**
  - Partition index is per texel
  - Colour endpoint mode is per partition (potentially all needed)
- **Weight data**
  - Bit selection shared
  - Adjacent texels share weight values for bilinear interpolation
    - Instead of 4 texels·4 interpolant·2 weight =32 weight values, only 3·3·2=18 are needed



Sharing

For a 2×2 request

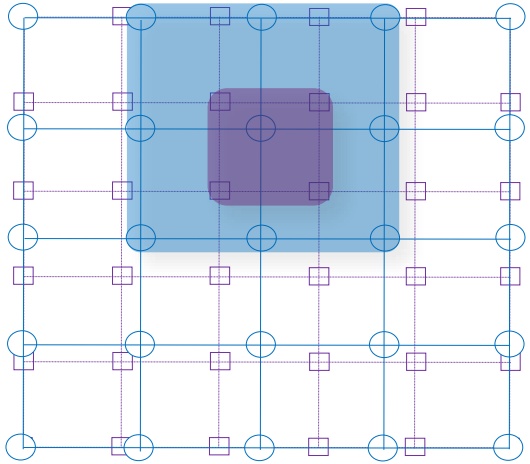
- **Configuration data is mostly shared for all texels**
  - Partition index is per texel
  - Colour endpoint mode is per partition (potentially all needed)
- **Weight data**
  - Bit selection shared
  - Adjacent texels share weight values for bilinear interpolation
    - Instead of 4 texels·4 interpolant·2 weight =32 weight values, only 3·3·2=18 are needed



Sharing

For a 2×2 request

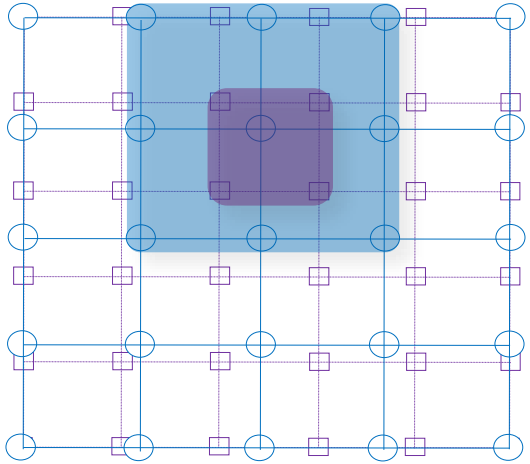
- **Configuration data is mostly shared for all texels**
  - Partition index is per texel
  - Colour endpoint mode is per partition (potentially all needed)
- **Weight data**
  - Bit selection shared
  - Adjacent texels share weight values for bilinear interpolation
    - Instead of 4 texels·4 interpolant·2 weight =32 weight values, only 3·3·2=18 are needed
- **Colour data**
  - Bit selection is shared
  - Limit of 18 colour values instead of 4 values · 4 partitions · 2 endpoints = 32
  - Some modes are not legal for 3 or 4 partitions



Sharing

For a 2×2 request

- **Configuration data is mostly shared for all texels**
  - Partition index is per texel
  - Colour endpoint mode is per partition (potentially all needed)
- **Weight data**
  - Bit selection shared
  - Adjacent texels share weight values for bilinear interpolation
    - Instead of 4 texels·4 interpolant·2 weight =32 weight values, only 3·3·2=18 are needed
- **Colour data**
  - Bit selection is shared
  - Limit of 18 colour values instead of 4 values · 4 partitions · 2 endpoints = 32
  - Some modes are not legal for 3 or 4 partitions
- **Interpolation**
  - Is independent (at least up to 4 texels)



Area

For a 2×2 request

- **Synthesis of four single-output decoders**
  - About $100 \cdot 10^3$ µm^2 in 40nm at 400MHz
- **Synthesis of one four-output decoders**
  - About $60 \cdot 10^3$ µm^2 in 40nm at 400MHz

- **Area savings is 40%**
  - Roughly corresponds to expected savings of:
    - 75% in configuration decoding
    - $1-{}^{18}/_{32} \approx 44\%$ in weight decoding
    - $1-{}^{18}/_{32} \approx 44\%$ in colour decoding
    - No savings in interpolating

Performance

For a 2×2 request

- **If not all texel request fall inside the encoded block there is a performance hit**
  - A 2×2 request at an edge requires two decoding cycles
  - A 2×2 request at a corner requires four decoding cycles

Performance

For a 2×2 request

- **If not all texel request fall inside the encoded block there is a performance hit**
  - A 2×2 request at an edge requires two decoding cycles
  - A 2×2 request at a corner requires four decoding cycles
- **Assuming each location is equally likely**
  - With footprint n×m
  - $\frac{(n-1)\cdot(m-1)}{n\cdot m}$ fall inside the block
  - $\frac{(n-1)+(m-1)}{n\cdot m}$ fall on an edge
  - $\frac{1}{n\cdot m}$ fall on a corner

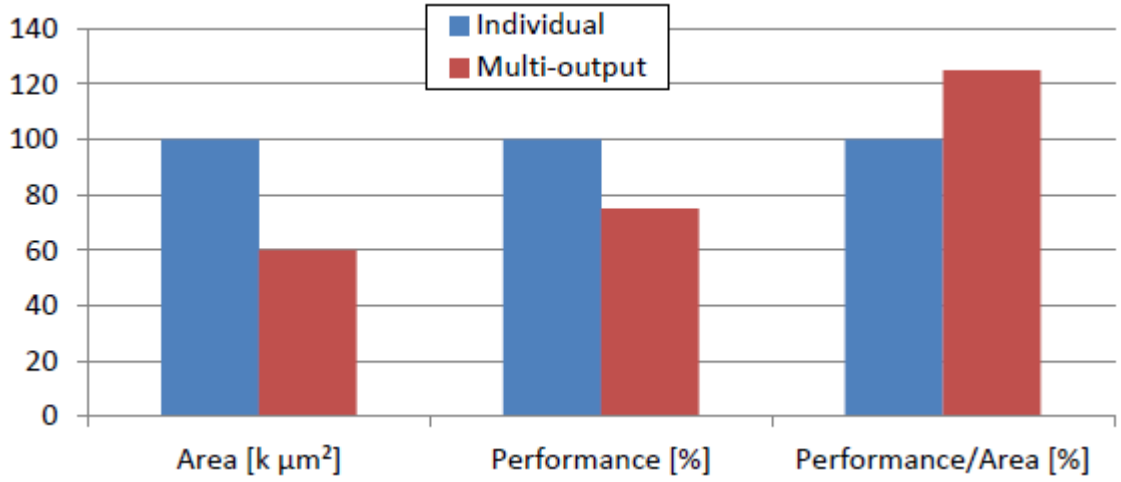
Table 1. MULTI-OUTPUT PERFORMANCE FOR EACH FOOTPRINT

4×4	5×4	5×5	6×5	6×6	8×5	8×6
64.0%	66.7%	69.4%	71.4%	73.5%	74.1%	76.2%
10×5	10×6	8×8	10×8	10×10	12×10	12×12
75.8%	77.9%	79.0%	80.1%	82.6%	83.9%	85.2%

Performance per area

For a 2×2 request

- **Area savings: 40%**
- **Average performance assuming each footprint is equally likely: 75.8%**
- **Performance/area improvement: 25% (75/60=1.25)**
  - Worst case: 7% (64.0/60), best-case: 42% (85.2/60)



Conclusion

- A multi-output decoder can decode a 2×2 block of texels for 40% less hardware cost than four individual decoders
- This is achieved by sharing common hardware, exploiting overlap in adjacent requests and exploiting the invalid space in the specification
- This comes at an performance cost of 25% on average (although smarter scheduling not discussed here can reduce this)
- The performance per area improves significantly by 25%

Questions?