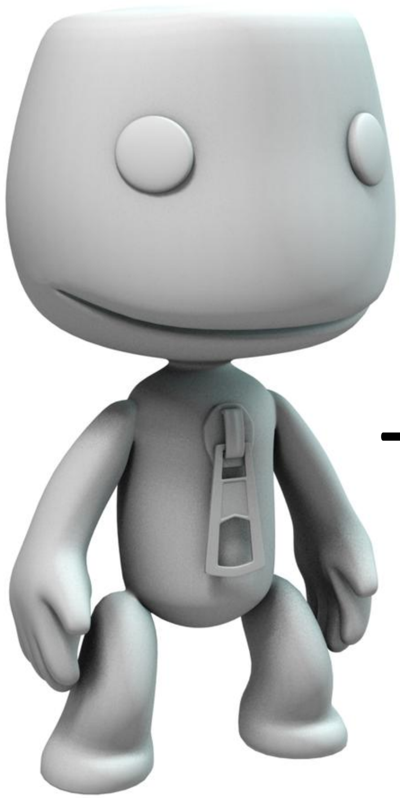
Texture Compression using Wavelet Decomposition

Pavlos Mavridis

Georgios Papaioannou

Texture Mapping

- Textures are 2D raster images that are *mapped* on 3D objects to add detail.



Model without Textures

+

2D Texture

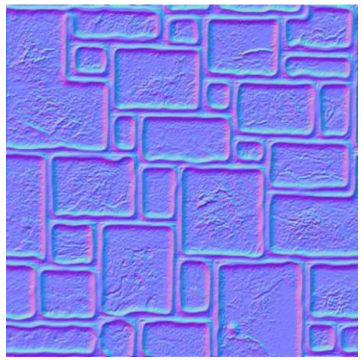
Texture Mapped Model

Source: blenderartists.org

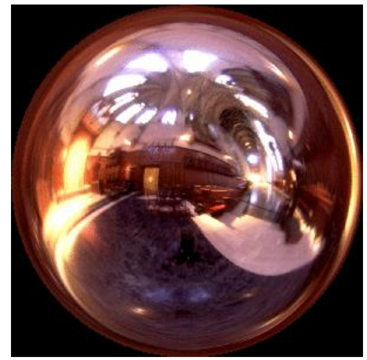
Texture Mapping

- Used excessively in computer graphics

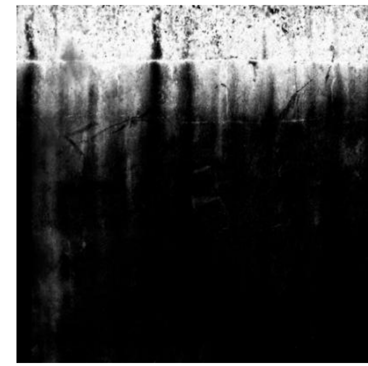
Albedo



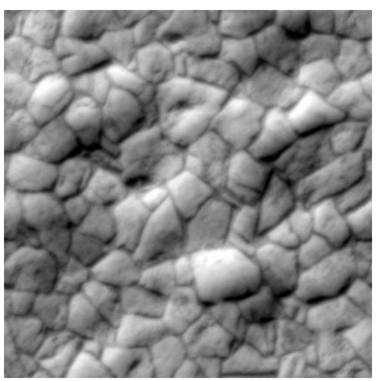
Normal Maps



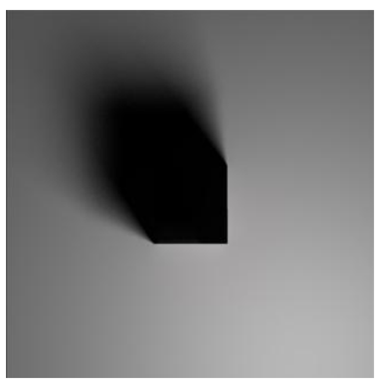
Environment Maps



Masks/DirtMaps/etc..



Displacement/Height



Light Maps

Each use-case has **varying requirements** on the quality and the number of channels.

Texture Mapping

- Texture Mapping is limited by:
  - **Bandwidth**
    - narrow memory bus on mobile hardware.
    - wider bus on desktops, but can be flooded by texture filtering and multiple texture layers.
  - **Storage space** (memory size is always limited)
- Solution: Texture Compression

Texture Compression

- Design Considerations [Beers et al. 96]
  - Fast decode
  - Fast Random Access
  - Can tolerate some loss of fidelity
  - Encoding time is not important (offline)
- Traditional image coding approaches (JPEG, …) **do not** guarantee these requirements
  (why? variable bit-rate / entropy encoding)

Traditional Image Coding

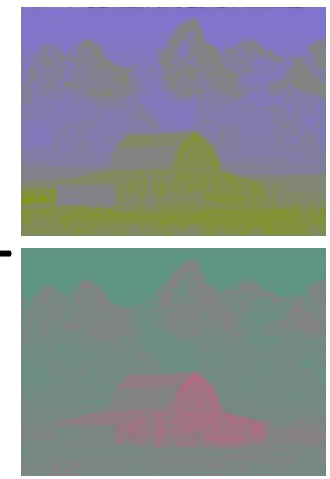
- Based on the following steps:
  1. Chroma Sub-sampling
  2. Energy compacting transform (DCT, DWT)
  3. Coefficient Quantization (with perceptual criteria)
  4. Coefficient Reordering
  5. Entropy encoding (RLE, Huffman, etc)

Chroma Sub-sampling

- The human visual system has finer *spatial* sensitivity to luminance than chrominance

RGB Input

Luminance



Down-Sampled Chrominance

A good down-sampling filter (Lanczos) should be used.

Chroma Sub-sampling

- Used for many years throughout the industry
  - Analog / digital TV broadcasting
  - JPEG and other image codecs
  - Blu-ray / DVD encoding
- No perceivable error:

Original – 24bpp

½ chroma – 12bpp

¼ chroma – 9bpp

Chroma Sub-sampling

- A lot of transforms for the luma / chroma decomposition
  - YCbCr (most popular)
  - YCoCg (better decorrelation [Malvar et al. 2003])

$$\begin{bmatrix} Y \\ C_o \\ C_g \end{bmatrix} = \begin{bmatrix} 1/4 & 1/2 & 1/4 \\ 1/2 & 0 & -1/2 \\ -1/4 & 1/2 & -1/4 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Additional bits are needed for the storage of YCoCg without loss of precision.
But we still have gain, since the spatial resolution of CoCg can be reduced.

YCoCg-R Color Space

- Similar to YCoCg
- Reduces the additional bit requirements to only 1-bit for Co and 1-bit Cg
  - Roughly 0.1dB gain (in PSNR) over YCoCg in our method

$$\begin{aligned} Co &= R - B \\ t &= B + (Co >> 1) \\ Cg &= G - t \\ Y &= t + (Cg >> 1) \end{aligned} \quad \Leftrightarrow \quad \begin{aligned} t &= Y - (Cg >> 1) \\ G &= Cg + t \\ B &= t - (Co >> 1) \\ R &= B + Co \end{aligned}$$

Assumes integer data. When using floating point textures, we have a small additional overhead to convert to integer in the shader.

Energy Compacting Transforms

- Most used transforms: DCT or DWT
  - We will focus on the second here

The 1-level DWT transform

Number of input pixels = Number of output coefficients (thus no compression yet)

The Haar Transform

- For every 2x2 block of texels apply this transfrom:

$$\begin{pmatrix} LL & HL \\ LH & HH \end{pmatrix} = \frac{1}{2} \begin{pmatrix} a+b+c+d & a-b+c-d \\ a+b-c-d & a-b-c+d \end{pmatrix}$$

- To get back the original data we apply **the same** transform:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} = \frac{1}{2} \begin{pmatrix} LL+HL+LH+HH & LL-HL+LH-HH \\ LL+HL-LH-HH & LL-HL-LH+HH \end{pmatrix}$$

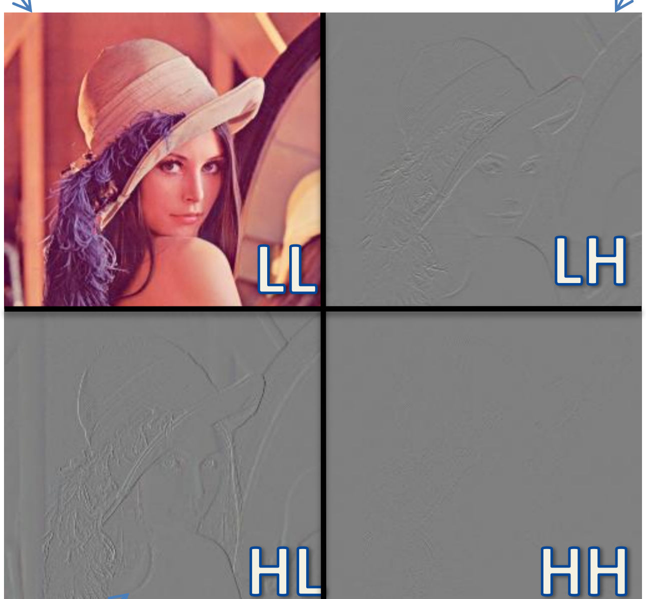
Decoding a 2x2 block (or a single pixel) requires 4 coefficients.

Coefficient Quantization

- For **lossy** compression the coefficients are quantized based on perceptual metrics

LL: Important

LH: Less Important



Low frequencies are more important.
Higher frequencies are less important.

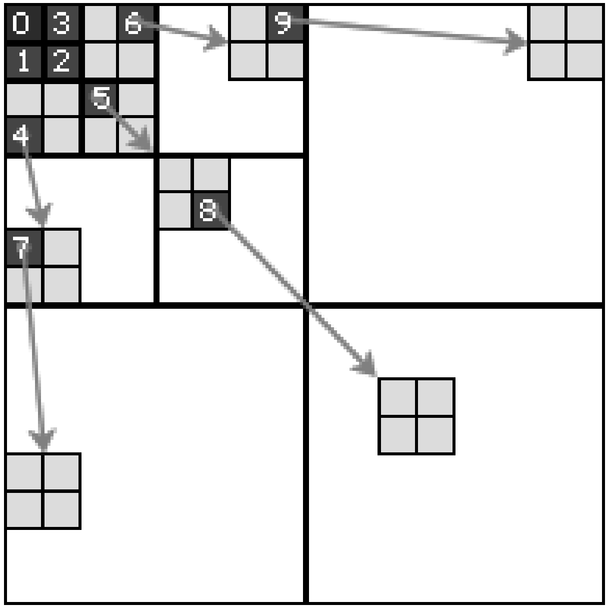
HL: Less Important

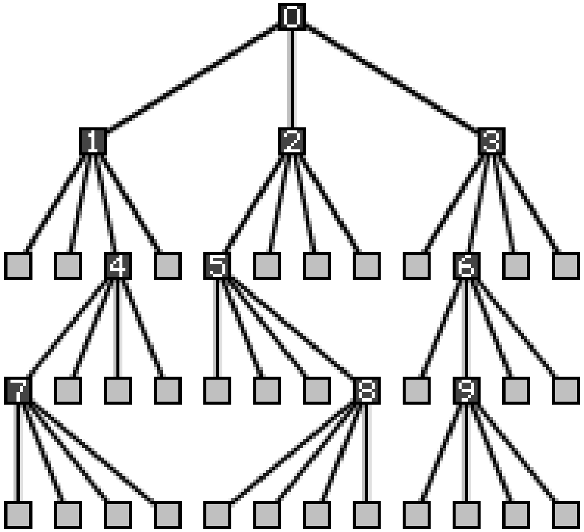
HL: Least Important

Coefficient Reordering

- Zero-trees [Shapiro 93]

Purpose: gather together coefficients with similar values





GPU adaptation [Boulton 2008]: Encode the tree as a texture, skip the entropy encoding to make it practical:

- Not fast: tree-traversal is required, bad for bandwidth
- The already available hardware is wasted
- Still might be useful for *wavelet compression of SH data.*

Entropy Encoding

- Lossless (RLE, Huffman, etc)
- Variable bit-rate
  - not good for random access
- Decoding entropy encoded data is inherently serial in nature
  -> No random acceeess

  -> **Not suited** for Texture Compression

So what TC methods do?

Previous Work on TC

- Mainly based on Quantization Approaches
  - **Global Codebook**
    (Color Palettes, Vector Quantization)
  - Or divide the image in blocks and use a smaller **Local Codebook** for each block
    S3TC/DXTC, BPTC, ETC and most modern texture compression formats

Global Codebooks

- **Color Palettes**
  - Replace each pixel/color with index into codebook
  - Cons: low compression rate / indirection
- **Vector Quantization** [Beers et al. 96]
  - Replace a block of pixels (2x2 or 4x4) with index
  - Used in the Dreamcast console (1998)
  - Indirection + large codebooks makes caching inefficient

So, the industry has moved to local codebooks…

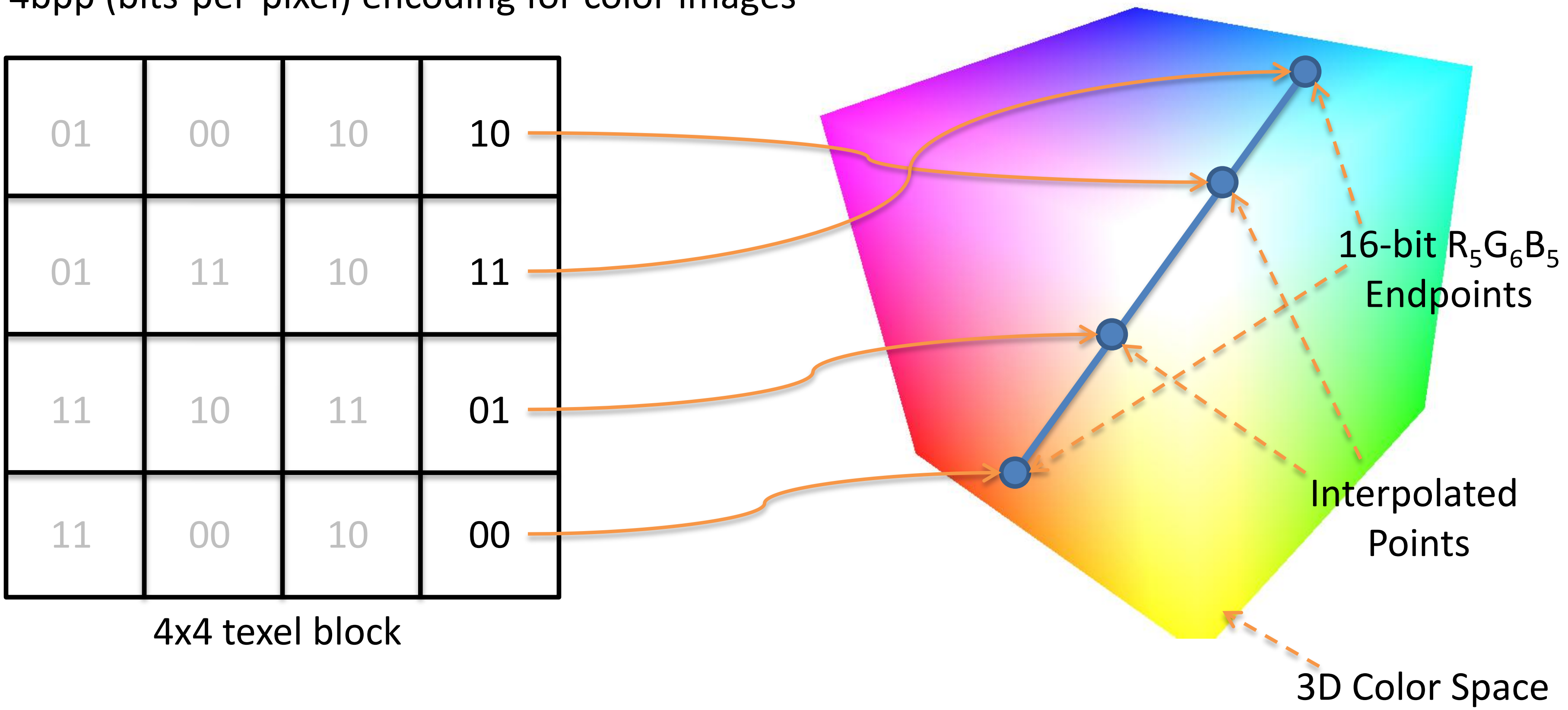
Block Compression with Local Codebooks

- The same quantization principle is applied on each 4x4 block of the image independently.
- *Local Codebook*: select some representative values from the local color space of the block.
- Texel values are given by indexing/interpolating the values in the local codebook.
- Characteristics:
  - No memory indirection
  - Each block is independent (both good and bad)
  - Fixed-rate

DXT1 Encoding

De-facto standard in desktop GPUs for more than a decade.

4bpp (bits-per-pixel) encoding for color images



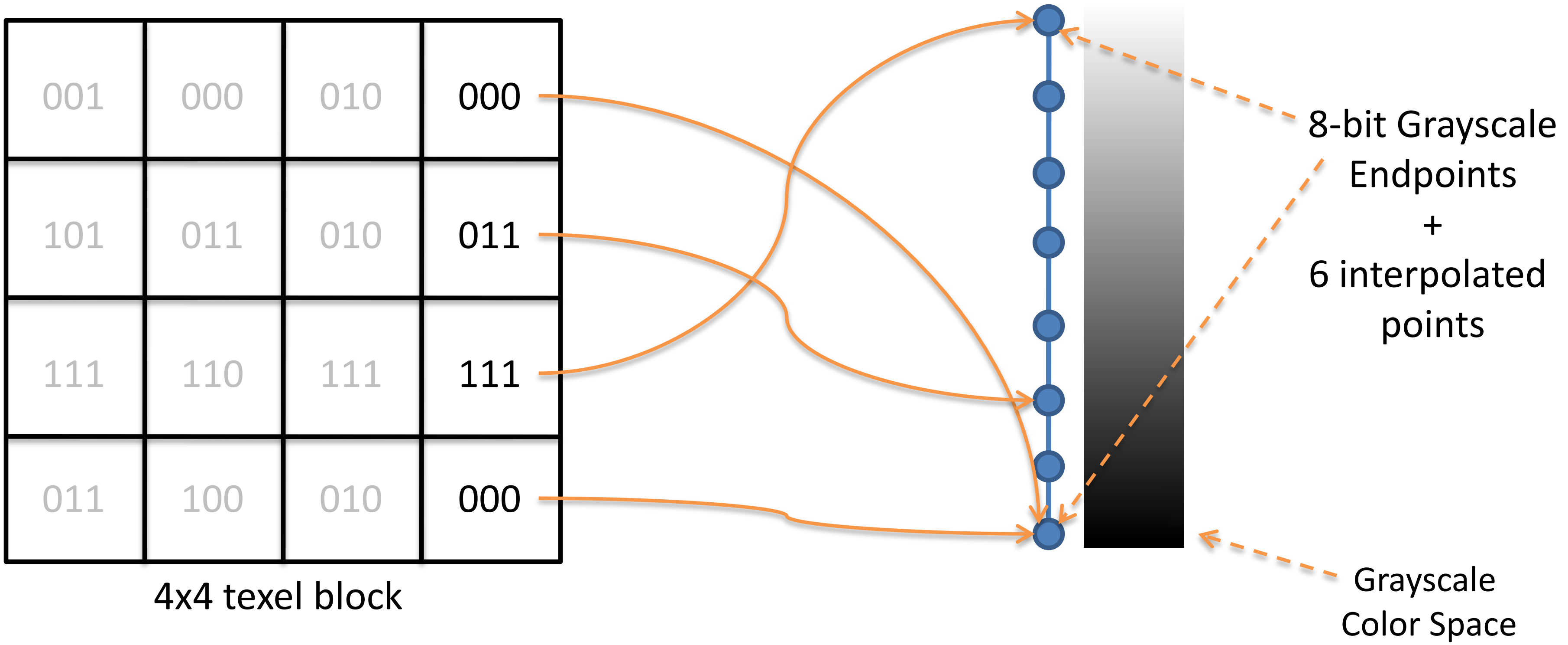
4x4 texel block

2bits index **X** 16 pixels **+** 16bits per endpoint **X** 2 endpoints = **64bits**

The same index is used for the three RGB values: assumes correlation!

DXT5/A Encoding

4bpp encoding for grayscale images



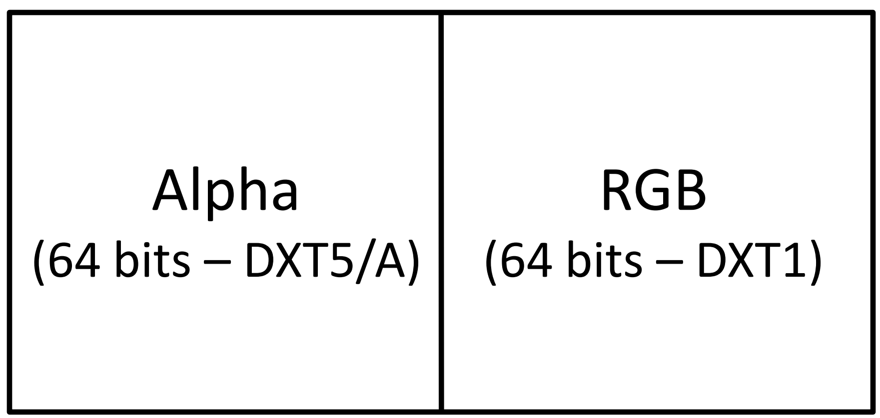
4x4 texel block

3 bits index **X** 16 pixels **+** 8 bits per endpoint **X** 2 endpoints = **64bits**

Color and grayscale images are encoded at the same rate!
And grayscale images have much more accuracy.

DXT5 Format

- Combines DXT1 for color and DXT5/A for alpha
- Alpha gets the preferential treatment
  (and we are going to exploit that later)



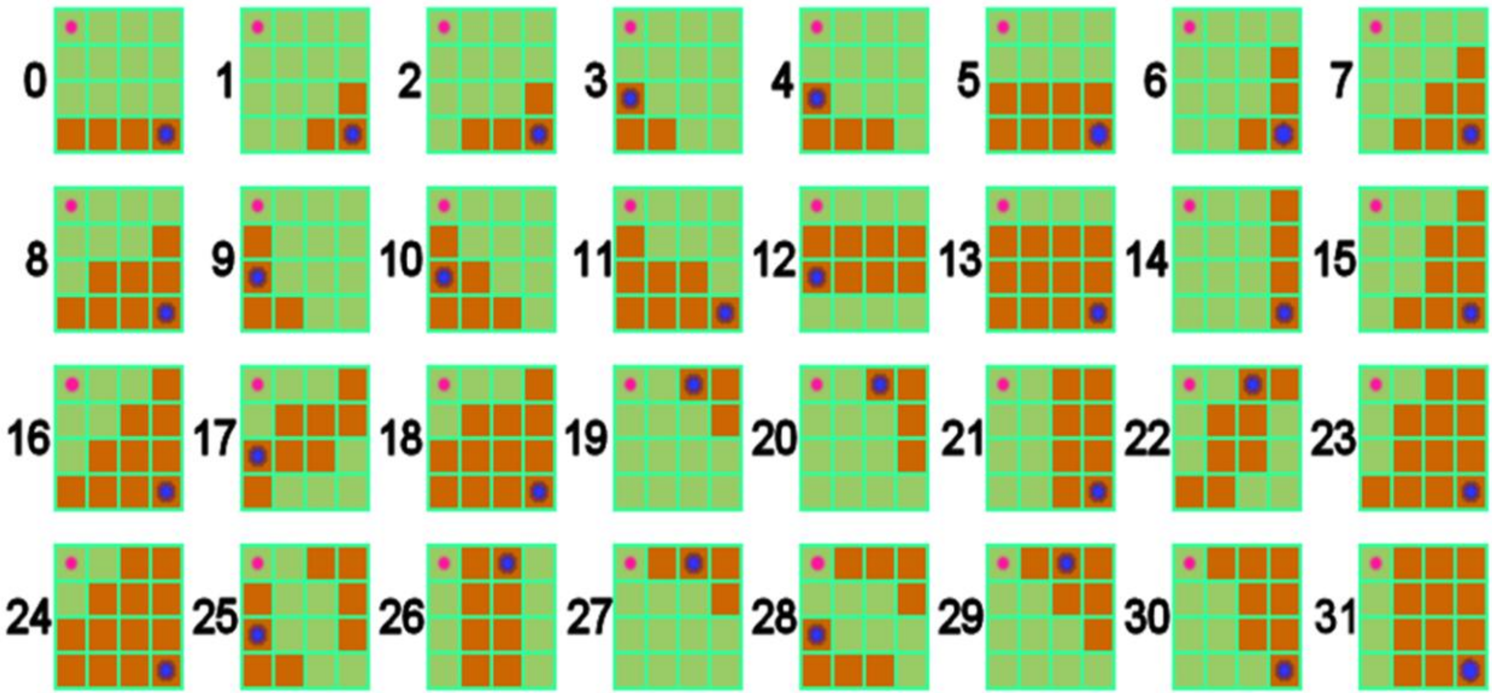
DXT5 Format
(encodes a 4x4 RGBA block at 128 bits)

BPTC / BC7 Encoding

- Available since OpenGL 4 / DX11
- Improves on DXT1 by defining partitions inside the 4x4 blocks
- Each partition has a unique endpoint pair
- Different number of partitions per block:
  - Blocks with less variance:
    - one partition, high precision endpoints
  - Blocks with more variance:
    - Up to 3 partitions, less precise endpoints

BPTC Partitions

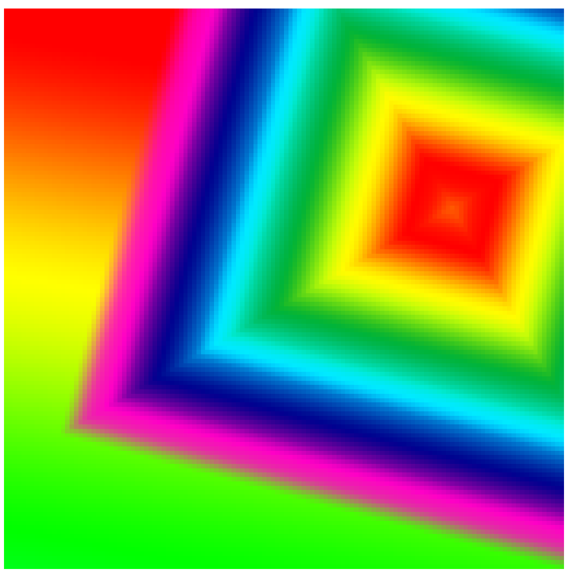
- Partitions are chosen from a palette of 64 predefined and well-chosen partition patterns



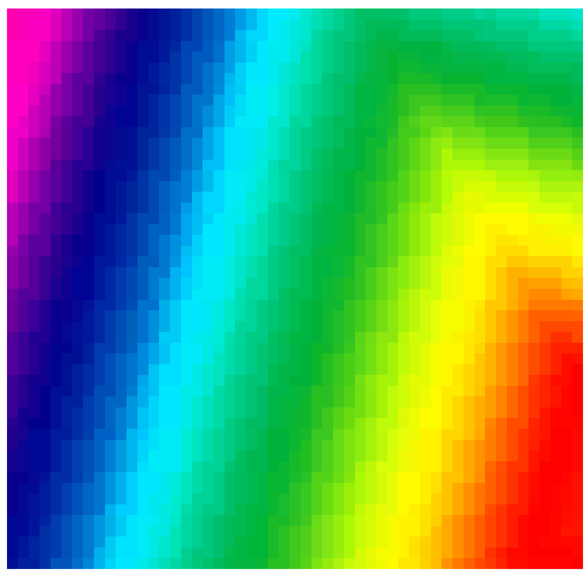
Source: Nvidia Corporation

BPTC / BC7 Encoding

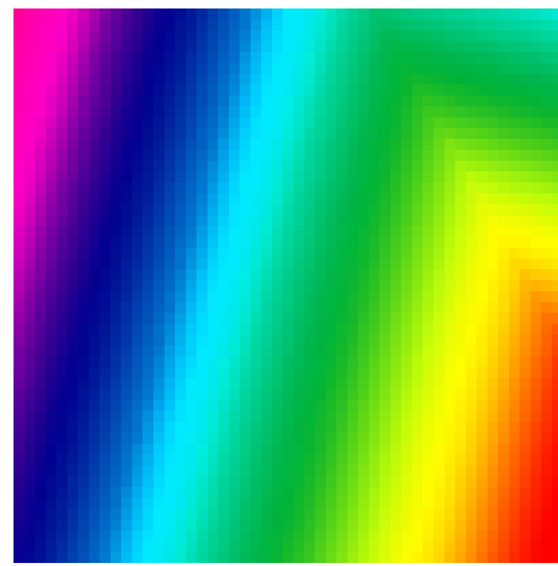
- 8 encoding modes: 4 RGB and 4 RGBA
  - Up to 8 points on the color interpolating line.
- 8 bpp rate (double the rate of DXT1)



Uncompressed – 24bpp
(128x128 pixels)



DXT1 – 4bpp
(4x Zoom)

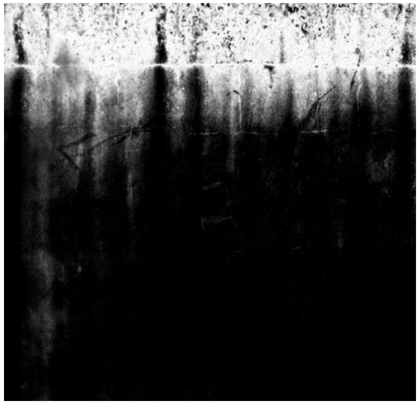


BC7 – 8bpp
(4x Zoom)

What’s wrong with (DX)TC today

- Very limited flexibility on bitrates
  - Color images: 4bpp encoding (OpenGL 4 adds the 8bpp BC7 format)
  - Gray scale images: also 4bpp encoding!
- Cannot fine-tune the size/quality tradeoff

Color Texture
4bpp DXT1



Grayscale *“Dirtmap”*
4bpp DXT5/A

Also we cannot go lower than 4bpp.

In DXTC, color and grayscale textures are encoded at the same bit-rate. Not always what we want.

What do we want (Motivation)

5 bpp | 3 bpp | 2.25 bpp | 1.75 bpp | 2 bpp

- More flexibility on bitrates
  - for both color and grayscale data
- Bonus points: Rather efficient implementation on existing hardware

Observation

- The TC methods largely ignore some of the standard image coding concepts
  (transform coding, chroma sub-sampling*)
- Is this the best choice?
  Perhaps it has been investigated in the past, but not documented.

Good opportunity for research!

**With the exception of ETC, which uses chroma sub-sampling but no transform coding*

Main Idea

LL: Important

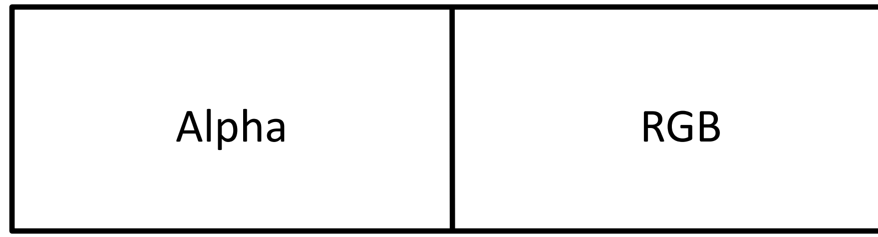
LH: Less Important

HL: Less Important

HL: **Least** Important

1-level HAAR decomposition

DXT5 Format



Coefficient Packing

LL: Important

LH: Less Important

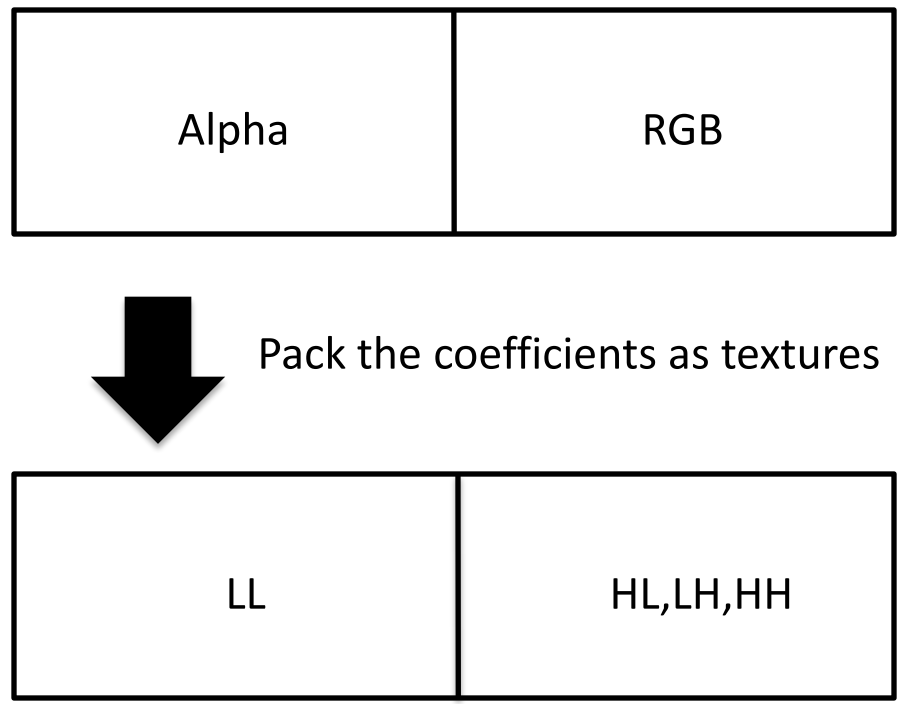
HL: Less Important

HL: **Least** Important

1-level HAAR decomposition

(can be seen as the equivalent to the re-ordering step in transform-coding)

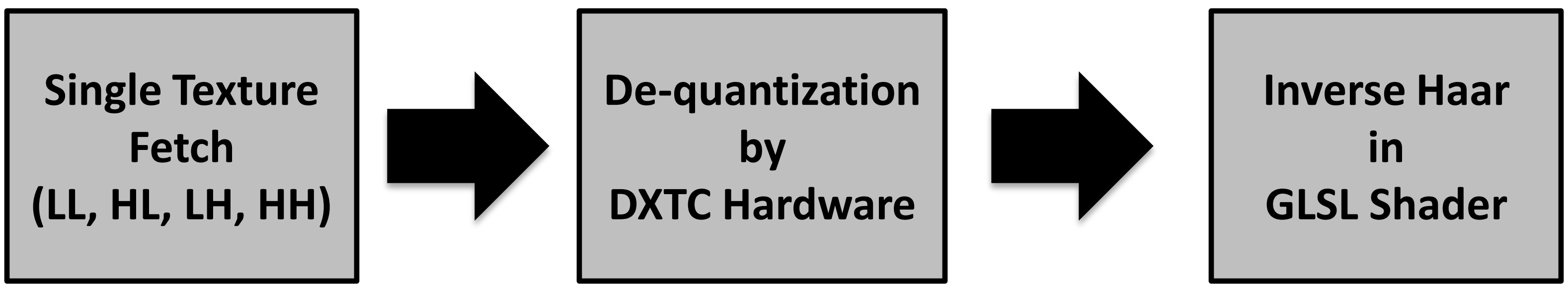
DXT5 Format

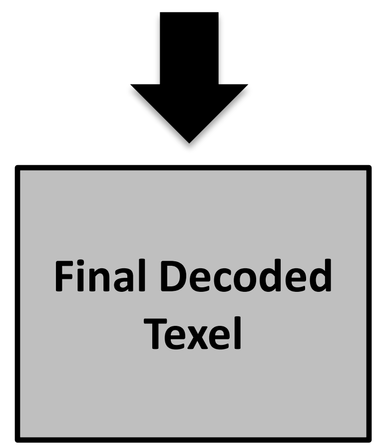


Original data: 8 bpp grayscale
Compressed : **2 bpp**
(4:1 fixed compression ratio)

Decoding

- Decode with a **single fetch**
  - Avoids the tree-traversal in the previous approaches





This is for the grayscale case. For color we must do this on each channel.

Bonus: The LL coefficient is the smaller mip level, so trilinear filtering is free!

First Results

- Looks rather good…

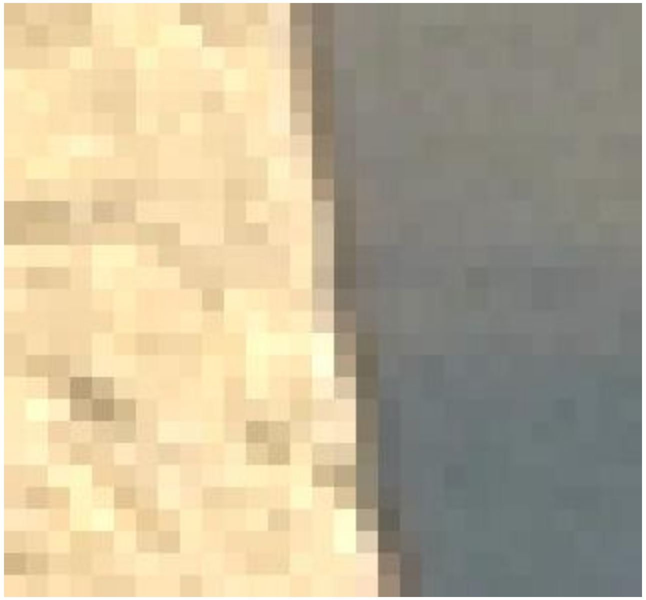
Original (24 bpp)

Compressed (3bpp)

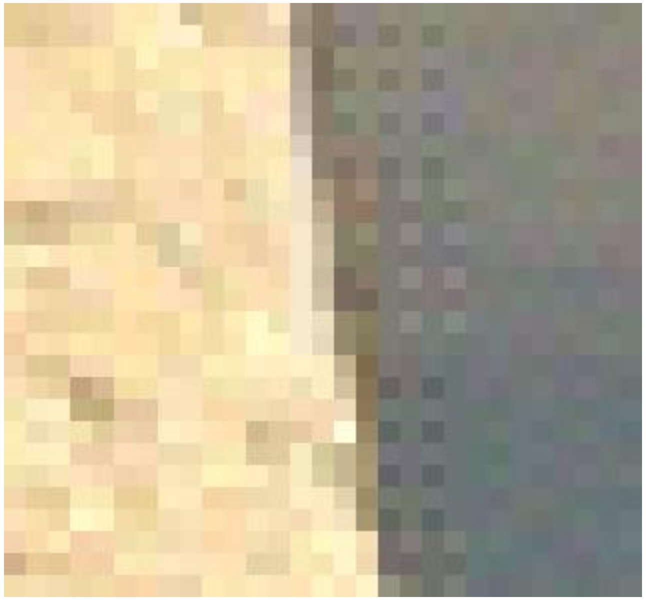
Color is encoded with 2:1 chroma sub-sampling in the YCoCg-R space

First Results

- Until you zoom in even further
  - 29x27 pixels at roughly 11x zoom:



Original



Compressed – 3 bpp

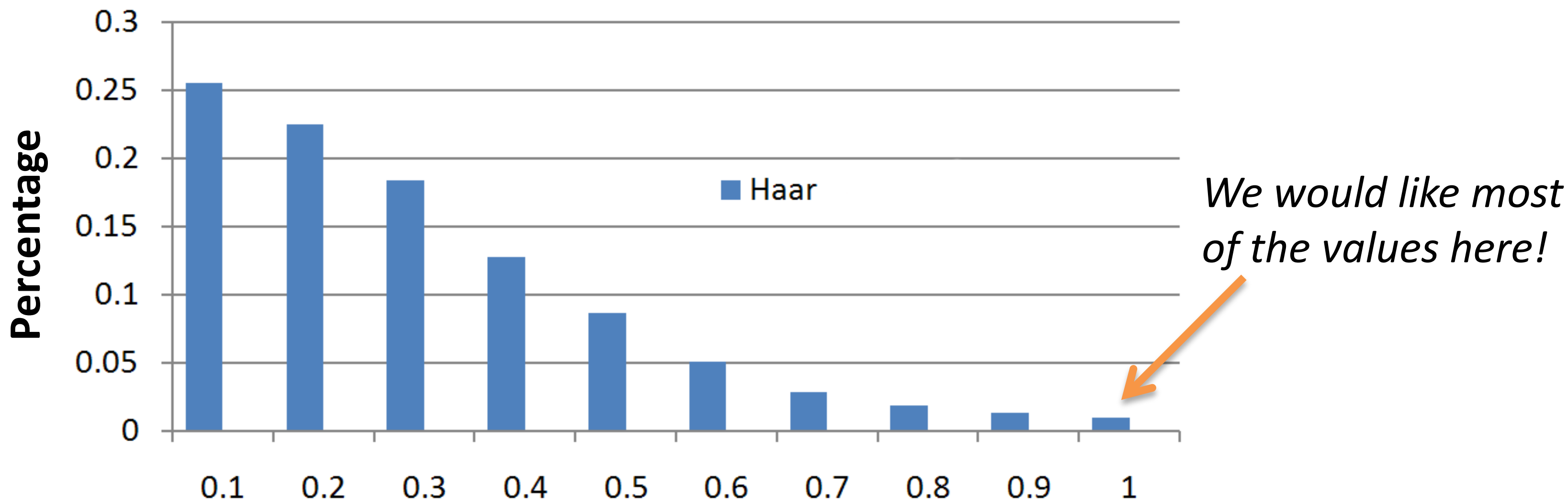
Blocky artifacts on sharp edges!

Reasons for this failure

- **Reason 1**: Wavelet coefficients are not correlated (but DXTC expects correlated data)
- **Reason 2**: Poor quantization of wavelet coefficients

Coefficient Correlation

- When the R, G and B components are not correlated, DXTC performs poorly
- *Haar* has well known de-correlation properties



Absolute *correlation coefficient*

for every pair of 4x4 blocks being encoded in the RGB channels

(data from the Lena image)

Our Solution

- How can we add some correlation back to the wavelet coefficients?
- We start with the inverse *Haar:*
  (transforms the coefficients back to the spatial domain, where they have rather good correlation)

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} = \frac{1}{2} \begin{pmatrix} LL + HL + LH + HH & LL - HL + LH - HH \\ LL + HL - LH - HH & LL - HL - LH + HH \end{pmatrix}$$

But if we encode the coefficients in the spatial domain (thus skipping HAAR), we will lose the advantage of having more information in LL, so we cannot use DXT5.

Solution: We keep LL and invert only the HL, LH, HH bands.

Partially Inverted Haar (PI - Haar)

- Instead of (LH, HL, HH) we define three new coefficients:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} = \frac{1}{2} \begin{pmatrix} LL + \underbrace{HL + LH + HH}_{HL'} & LL \underbrace{- HL + LH - HH}_{LH'} \\ LL + \underbrace{HL - LH - HH}_{HH'} & LL - HL - LH + HH \end{pmatrix}$$

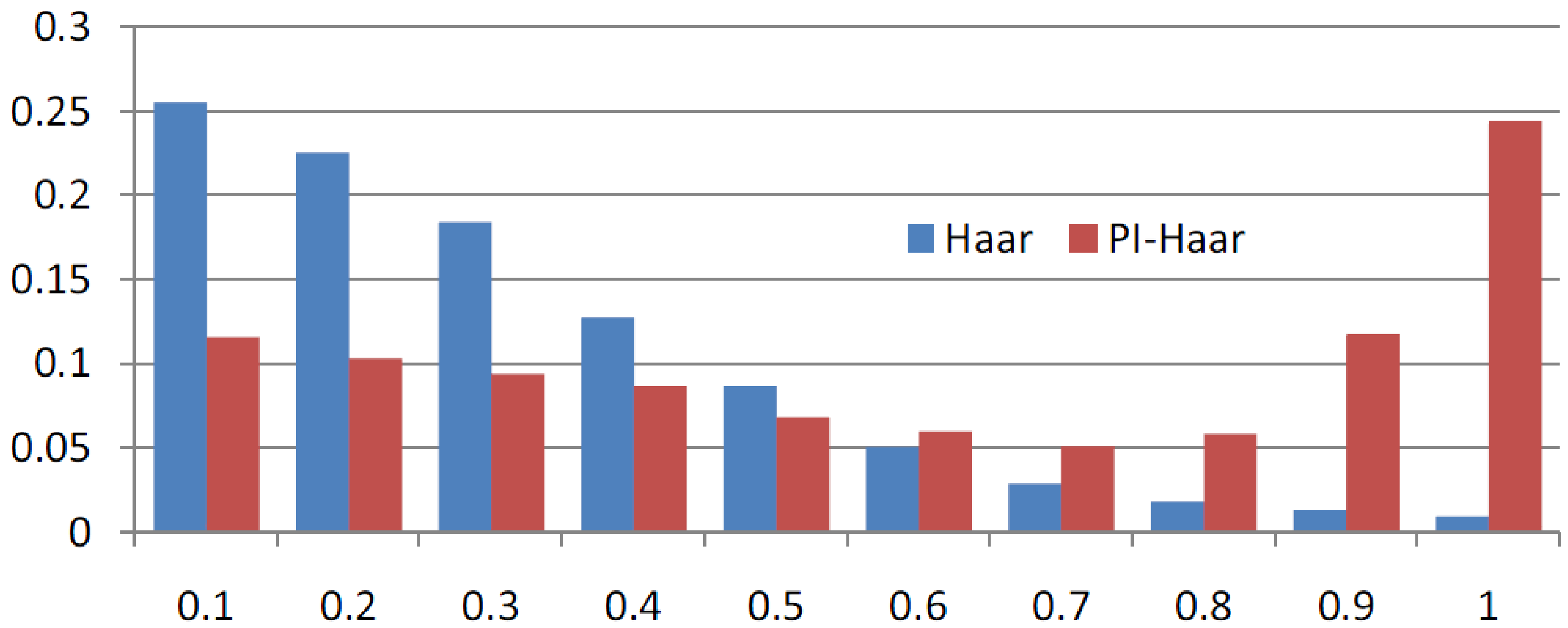
This can be derived from HL', LH' and HH'

- We also add a weight (w) to limit the influence of HH:

$$\begin{pmatrix} HL' \\ LH' \\ HH' \end{pmatrix} = \begin{pmatrix} 1 & 1 & w \\ -1 & 1 & -w \\ 1 & -1 & -w \end{pmatrix} \begin{pmatrix} HL \\ LH \\ HH \end{pmatrix}$$

We call the above transform ***Partially Inverted Haar*** *(PI - Haar)*

Histogram of Correlations



Still not perfect, but with the new transform we have more values towards 1

How much improvement?

Compression Error of the Red channel

Transform	R	G	B	MSE_r
Haar	HL	0	0	4.6
	HL	LH	HH	31.1
PI- Haar	HL’	0	0	4.3
	HL’	LH’	HH’	11.0

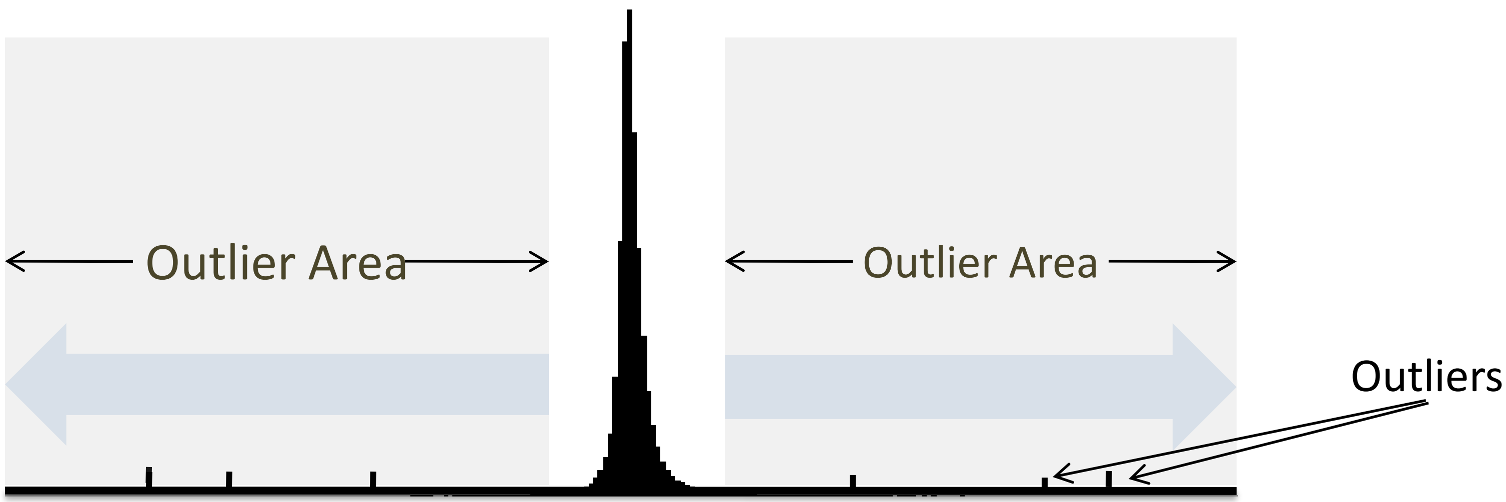
Compressing only the R channel (BG->black), the error is small.
Adding the BG channels, the error gets 6 times higher.

PI-Haar coefficients show better compressibility (better correlation)

(Data from the Lena image)

Coefficient Quantization

- Most coefficients are clustered towards zero
  - They will be quantized to the same value
  - The available spectrum is not used efficiently
- Some coefficients still exist at the edges of the spectrum
  - Statistical ***outliers*** from very sharp features on the original image



Histogram of Wavelet Coefficients

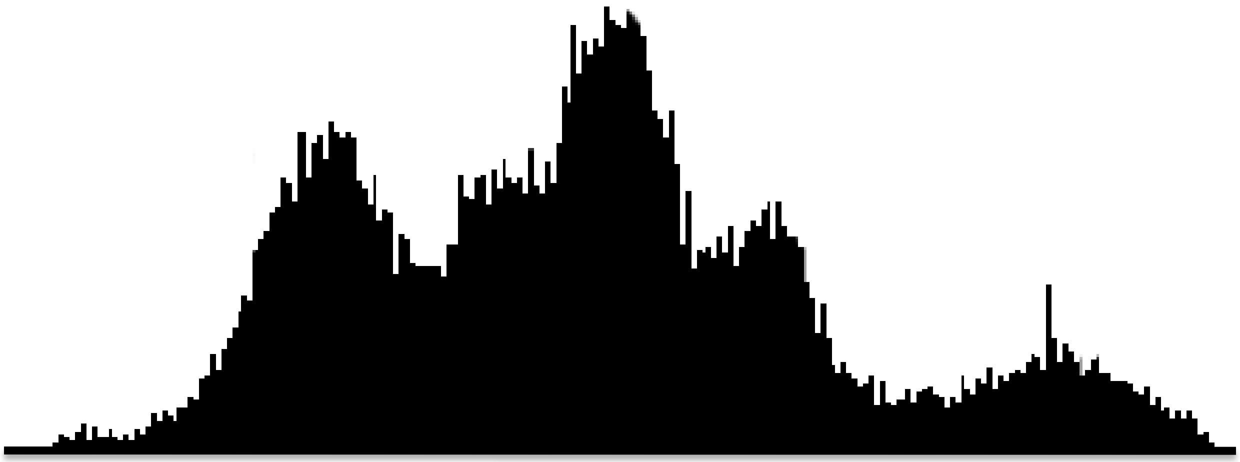
(Data from the Lena image)

Coefficient Quantization

- **Solution:** Clamp the outliers and normalize.
- An exponential scale to the coefficients also helps to evenly redistribute the values
  - but the gains are rather minimal
  - makes decoding more expensive
    (justified only if the highest possible quality is required)

Coefficient Quantization

- An optimization process (brute force) decides how much outliers to cut (and the optimal gamma space)
- After optimization:



Much better distribution of values and better use of the available spectrum.

For decoding, after fetching the coefficients we scale them back to their original range.

One more Optimization

- Use BPTC / BC7 instead of DXT5
- Similar PSNR with DXT5 but less artifacts, because the wavelet coefficients are handled better.

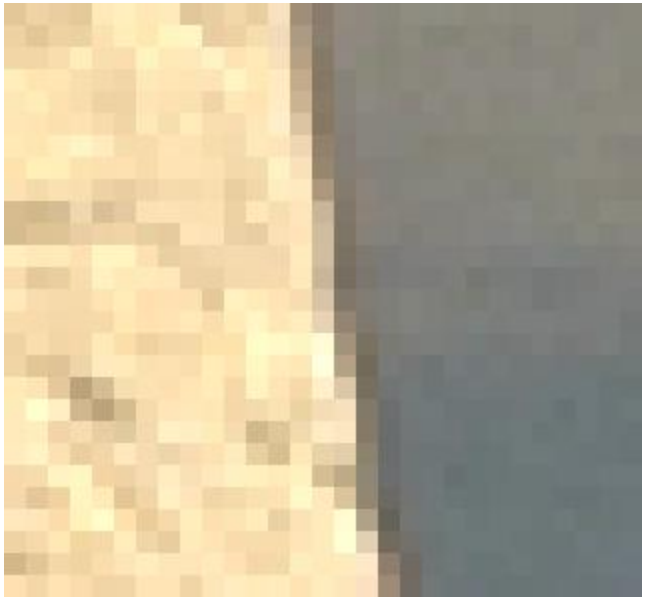
BC7 Packing:

R	G	B	→ Channels of a BC7 texel
LH’	LL’	HL’	→ Coefficients
25%	50%	25%	→ Weights during quantization

But we have to drop completely the HH’ coefficients.

New Results

- Now the artifacts are gone:

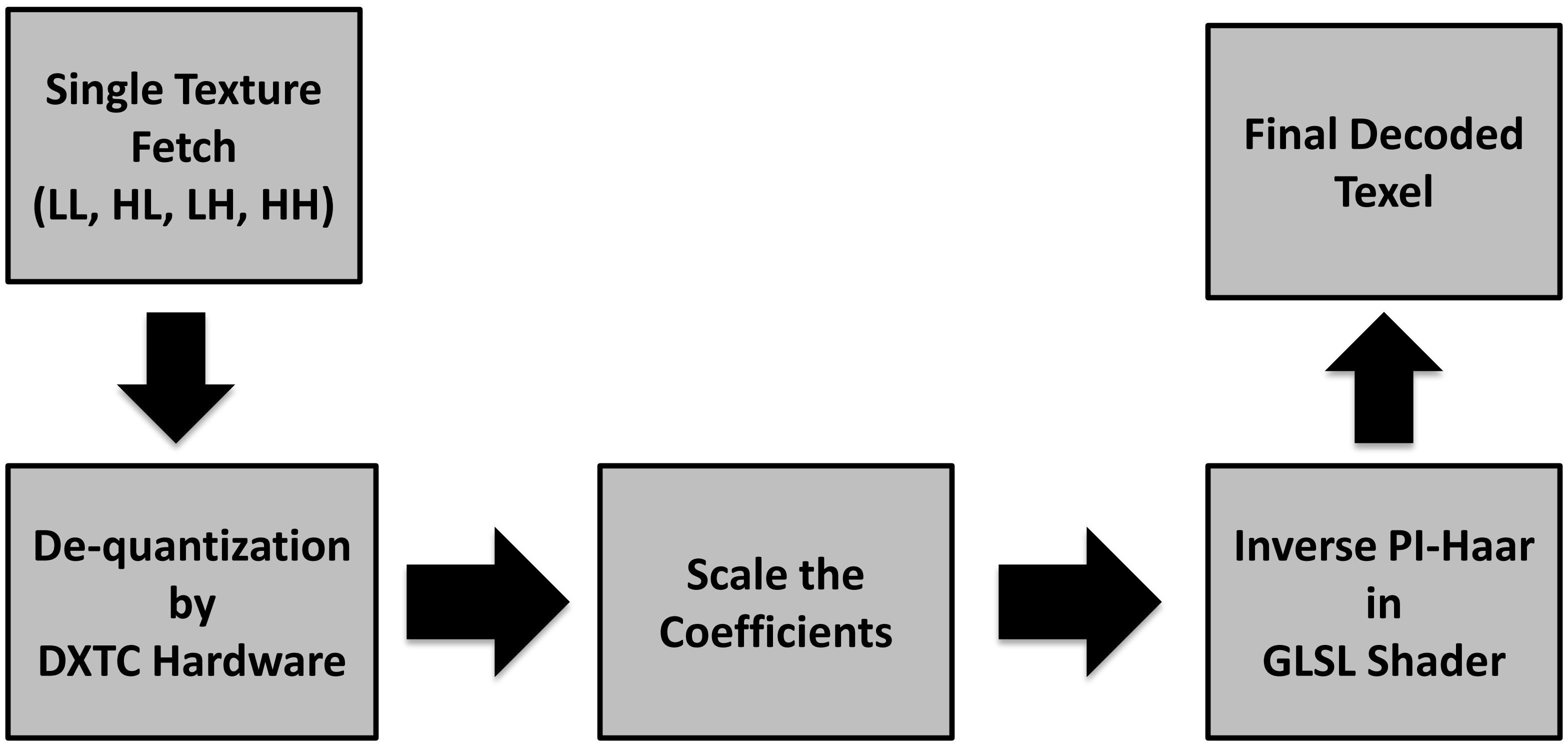


Original (24 bpp)

Compressed (3bpp)

Decoding (updated)

- Still decodes with a **single fetch**



Format table

- Combine different encodings to get new texture formats
  - More flexibility in bit-rate selection

Bit-rate	Luma	Chroma	Quality (PSNR)
5.0 bpp	DXT5/A	2:1 wavelet	High (+2.9dB)
4.0 bpp	DXT1		Baseline
3.0 bpp	wavelet	2:1 wavelet	Low (-3.0 dB)
2.25 bpp	wavelet	4:1 wavelet	Lowest (-3.8 dB)

Grayscale encoding: 2bpp wavelet format and 4bpp DXT5/A format.

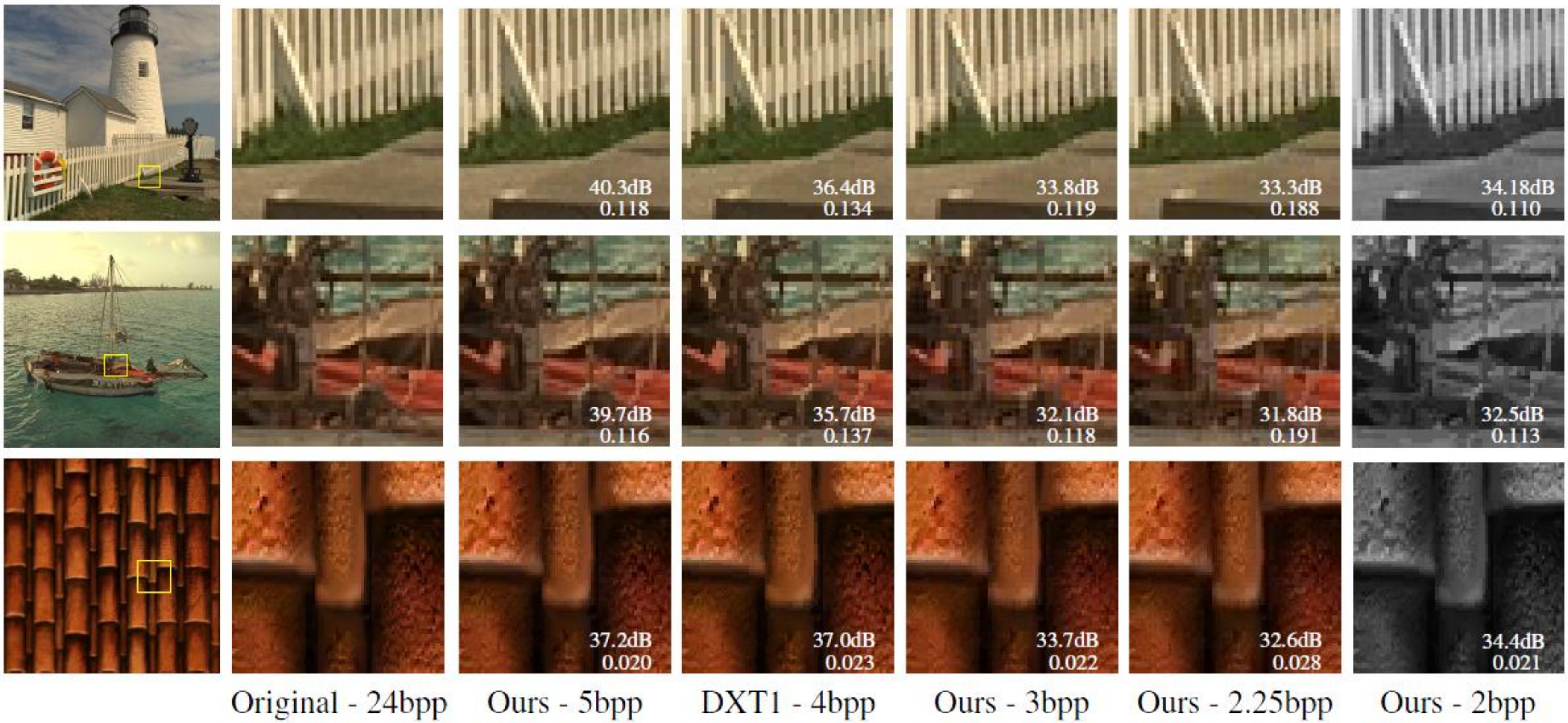
Test Dataset

- Kodak lossless image suite: standard benchmark for image coding algorithms
- 24 *representative* photos taken with a 3-CCD camera (no Bayer artifacts)

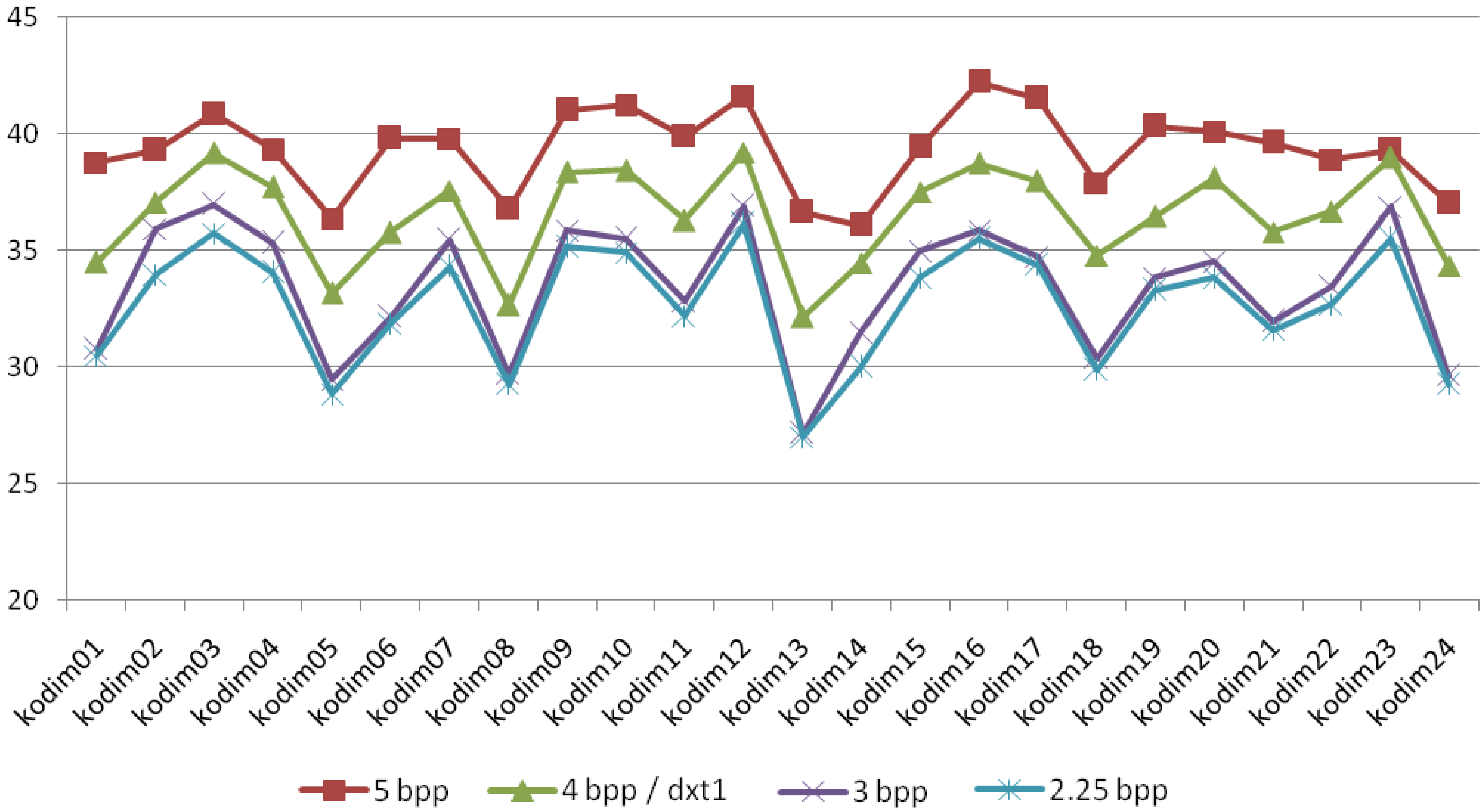





Results


40.3dB
0.118
36.4dB
0.134
33.8dB
0.119
33.3dB
0.188
34.18dB
0.110
39.7dB
0.116
35.7dB
0.137
32.1dB
0.118
31.8dB
0.191
32.5dB
0.113
37.2dB
0.020
37.0dB
0.023
33.7dB
0.022
32.6dB
0.028
34.4dB
0.021
Original - 24bpp
Ours - 5bpp
DXT1 - 4bpp
Ours - 3bpp
Ours - 2.25bpp
Ours - 2bpp

PSNR Results on Kodak



Comparison with Alternatives

- Is it better than using lower resolution textures to get the same gains?

Original | Scaled DXT1 (2.25 bpp) | Proposed (2.25 bpp)

As expected, high frequencies get blurrier.

For the 2.0bpp gray-scale format, the PSNR gain over scaled DXTC is 2.2dB
For the 2.25bpp color format, the PSNR gain over scaled DXTC is 1.4dB

Multilevel Decomposition

- The algorithm can be applied recursively on the LL coefficient
- We do not recommend this because:
  - Data will be scattered in memory
  - More complex (slower) decoding
  - Lower quality

(But we have still investigated this case for completeness)

Multilevel Decomposition

Original - 8bpp

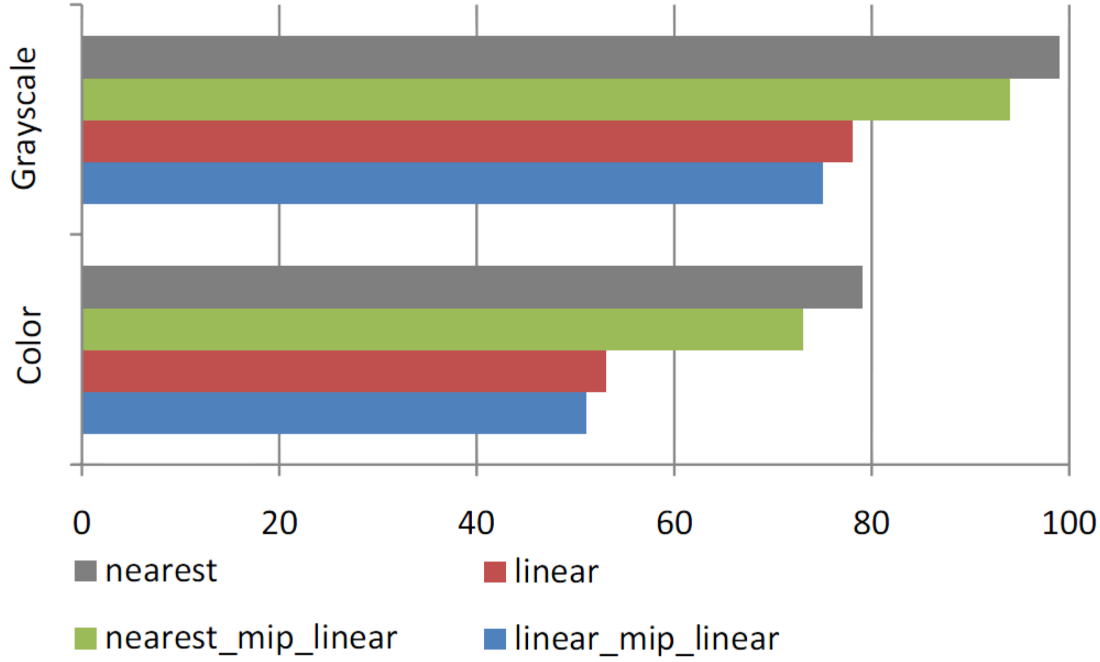
1 level - 2 bpp

2 levels - 1.5 bpp

Combined with chroma sub-sampling we can get a 1.75bpp RGB format.

Texture Filtering

- Filtering should happen after decompression
- Our method breaks hardware filtering
- Must perform filtering it in the shader



100% indicates the speed of the native hardware.

The overhead for the unfiltered grayscale case is almost zero!

Summary

- **Advantages**
  - Improved flexibility
  - Very simple decode
  - Takes advantage of existing hardware
  - Patent free!
- **Disadvantages**
  - Texture filtering has a performance hit
  - One texture unit per compressed channel

Concurrent Work (ASTC)

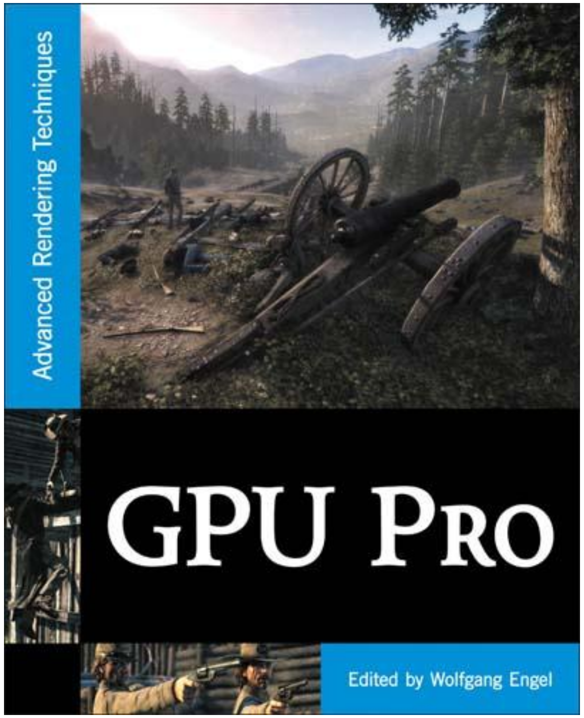
- The industry recognized the lack of flexibility
- ARM has proposed **ASTC**
  - Amazing work!
  - Bit-rates ranging from 0.89bpp up to 8bpp
  - It requires a new hardware implementation (while our method can be rather efficient on existing GPUs)
- Orthogonal to our approach:
  - Still does not uses chroma sub-sampling or any transform coding concepts.
  - **Future work:** Use ASTC to encode the wavelet coefficients in our framework

Future Work

- Other encoding formats for the wavelet coefficients
  - ASTC or even propose new encodings
- Investigate other image decomposition transforms.
- Extend the method for floating-point and volume data.

Future Work

- Frame Buffers consume a lot of memory too
  - HDR (half-float precision)
  - MSAA
  - "Retina" displays
- Frame Buffer compression
  - On existing GPUs!
- Upcoming article on GPU Pro 4
  (and under peer review for an academic journal)



Thank You!

- Questions?
- More info:
  - http://pmavridis.com
  - http://graphics.cs.aueb.gr

Title image kindly borrowed from SIGGRAPH Asia 2011

BACKUP SLIDES

Other Transforms

- Observation: Even without DXTC quantization, performing Haar or PI-Haar wirh 8-bit precision results in loss of quality
- Solution(?) : use **PLHaar** or **S-Transform** (variations of Haar to work on integers)
- Turns out these transforms give lower PSNR. (even if we partially invert them, with the same methodology)