# Texture Compression

Jacob Ström, Ericsson Research

# Overview

- Benefits of texture compression
- Differences from normal image compression
- Texture compression algorithms
  - Palettized textures
  - BTC
  - CCC
  - S3TC
  - PVR-TC
  - PACKMAN
  - ETC (Ericsson Texture Compression)
- Normal map compression
  - 3Dc

# why 3D graphics
## On a mobile phone

- Killer app: User Interfaces
- But also...
  - Games
  - Maps,
  - Browsing, Screen Savers, Messaging and more...

# Why is 3D Graphics Hard
## on a Mobile Phone?

Limited resources:

# Why is 3D Graphics Hard
## on a Mobile Phone?

Limited resources:

› Small amount of memory
› Little memory bandwidth
› Little chip area for special purpose
› Powered by batteries

# Texture Compression
## and the bus

BUS

Texels

GPU

Display

# Texture Compression Helps

- Small amount of memory
  - More texture data can fit in the limited amount of memory
- Little memory bandwidth
  - More texturing possible for same amount of bandwidth
- Little chip area for special purpose
  - A texture cache using compressed data can be made smaller
- Powered by batteries
  - Reduced bandwidth means lower energy consumption
- However, texture compression is also good for computers and games consoles!

# Texture Mapping is a Bandwidth Hog

- For each pixel drawn in the image, eight pixels from the texture (texels) are usually read.

drawn image

texture mipmap levels

# Texture Compression
## and the bus

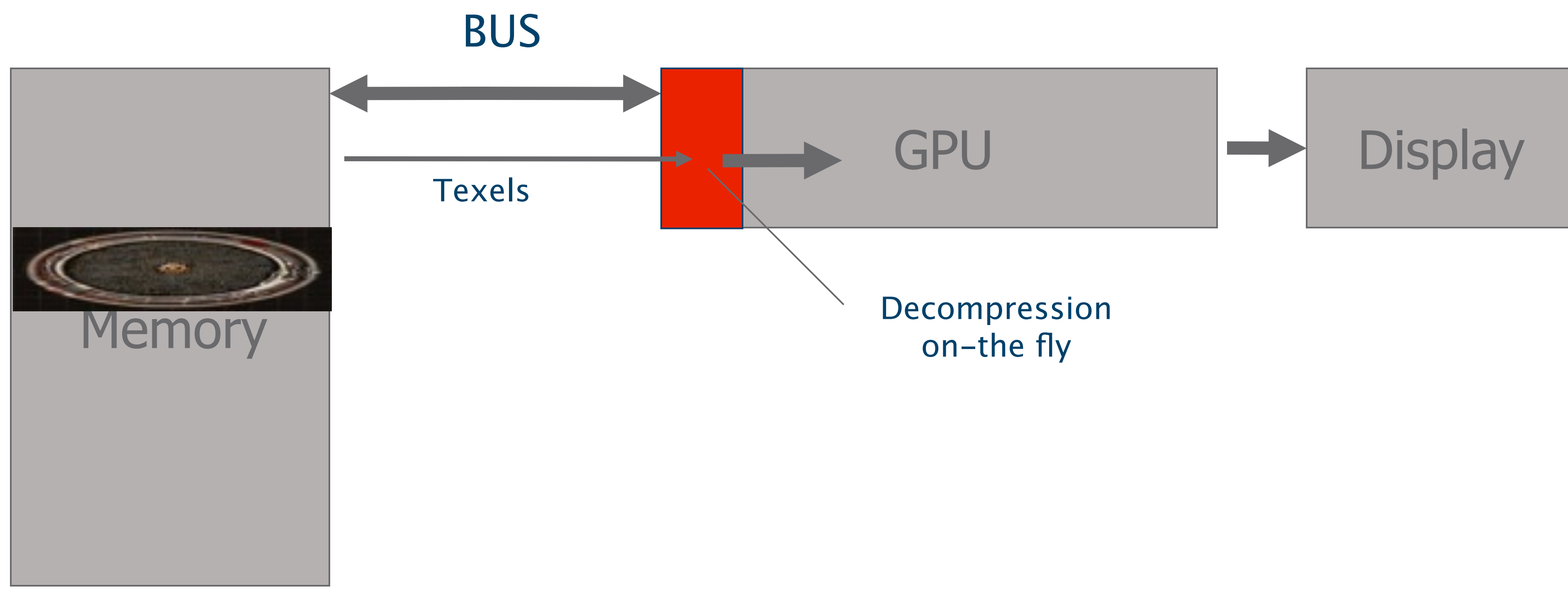
BUS

GPU

Display

Texels

# Texture Compression
## and the Bus



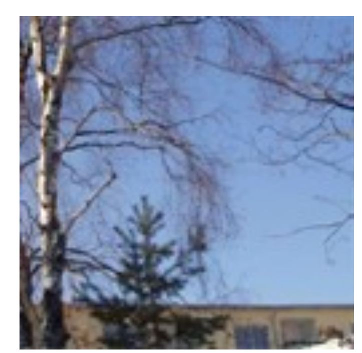
# Benefits of Texture Compression

- Higher Performance
  - Bandwidth is usually the factor limiting performance in rasterization-based graphics hardware.
  - Texture Compression reduces texturing bandwidth with a factor of up to 6
  - Spare bandwidth can be used for higher performance, or lower power consumption (mobile case)
- Higher Quality! (Yes, really…)
  - Even a huge video memory gets full.
  - With a compression ratio of 6, you can increase the resolution one mipmap level and still save memory

# Benefits of Texture Compression

- Higher Performance
  - Bandwidth is usually the factor limiting performance in rasterization-based graphics hardware.
  - Texture Compression reduces texturing bandwidth with a factor of up to 6
  - Spare bandwidth can be used for higher performance, or lower power consumption (mobile case)
- Higher Quality! (Yes, really…)
  - Even a huge video memory gets full.
  - With a compression ratio of 6, you can increase the resolution one mipmap level and still save memory

with texture compression, 128x128 pix, 8192 bytes



# Benefits of Texture Compression

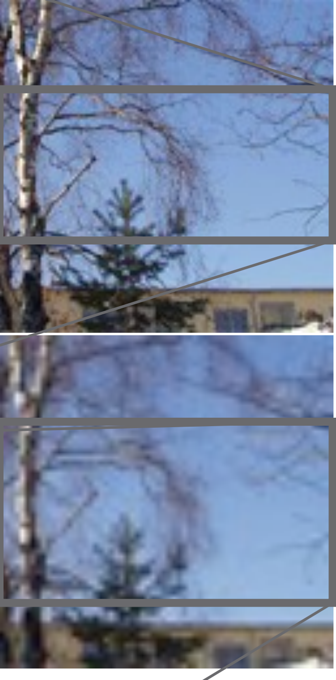
- Higher Performance
  - Bandwidth is usually the factor limiting performance in rasterization-based graphics hardware.
  - Texture Compression reduces texturing bandwidth with a factor of up to 6
  - Spare bandwidth can be used for higher performance, or lower power consumption (mobile case)
- Higher Quality! (Yes, really…)
  - Even a huge video memory gets full.
  - With a compression ratio of 6, you can increase the resolution one mipmap level and still save memory

with texture compression, 128x128 pix, 8192 bytes

no texture compression, downsampled to 64x64, 12288 bytes

# Benefits of Texture Compression

with texture compression,
128x128 pix, 8192 bytes

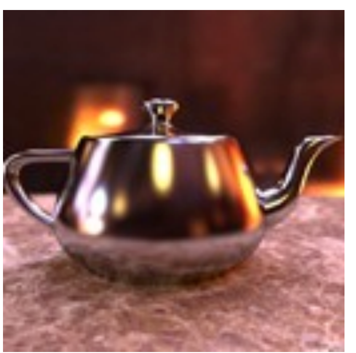


no texture compression,
downsampled to 64x64,
12288 bytes

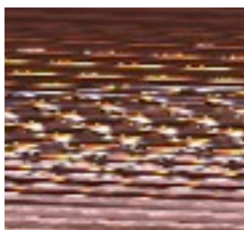
# Difference to Image Compression

## why not just use JPEG?

› First a short recap on how JPEG compresses images



image



JPEG bits

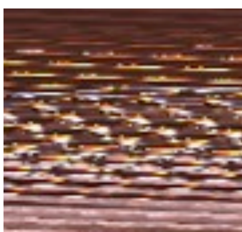
left image courtesy of Henrik Wann Jensen

# Difference to Image Compression

## why not just use JPEG?

- First a short recap on how JPEG compresses images
  - The image is first divided into 8x8 blocks.

image



JPEG bits

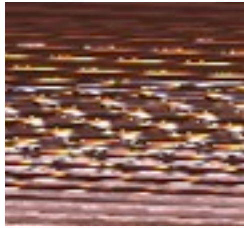
left image courtesy of Henrik Wann Jensen

# Difference to Image Compression

## why not just use JPEG?

- First a short recap on how JPEG compresses images
  - The image is first divided into 8x8 blocks.

image



JPEG bits

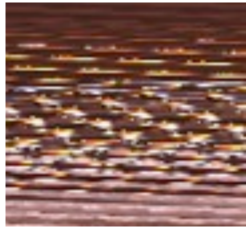
left image courtesy of Henrik Wann Jensen

# Difference to Image Compression

## why not just use JPEG?

- First a short recap on how JPEG compresses images
  - The image is first divided into 8x8 blocks.
  - Each block is then encoded and put into the file

image



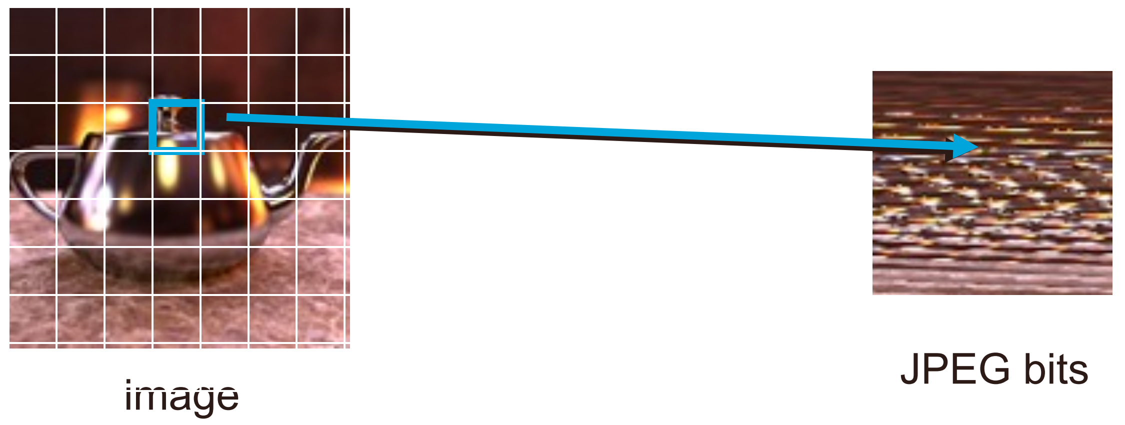
JPEG bits

left image courtesy of Henrik Wann Jensen

# Difference to Image Compression

## why not just use JPEG?

- First a short recap on how JPEG compresses images
  - The image is first divided into 8x8 blocks.
  - Each block is then encoded and put into the file




left image courtesy of Henrik Wann Jensen

# Difference to Image Compression

## why not just use JPEG?

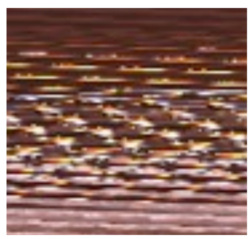
- Most image compression algorithms, such as JPEG, uses variable bit length coding (VLC).
- A block that is hard to code is allowed to occupy more bits than a block that is, for instance, just black.

# Difference to Image Compression

## why not just use JPEG?

- Most image compression algorithms, such as JPEG, uses variable bit length coding (VLC).
- A block that is hard to code is allowed to occupy more bits than a block that is, for instance, just black.

image



JPEG bits

left image courtesy of Henrik Wann Jensen

# Difference to Image Compression

## why not just use JPEG?

- Most image compression algorithms, such as JPEG, uses variable bit length coding (VLC).
- A block that is hard to code is allowed to occupy more bits than a block that is, for instance, just black.

left image courtesy of Henrik Wann Jensen

# Difference to Image Compression

## why not just use JPEG?

- Most image compression algorithms, such as JPEG, uses variable bit length coding (VLC).
- A block that is hard to code is allowed to occupy more bits than a block that is, for instance, just black.

left image courtesy of Henrik Wann Jensen

# Difference to Image Compression

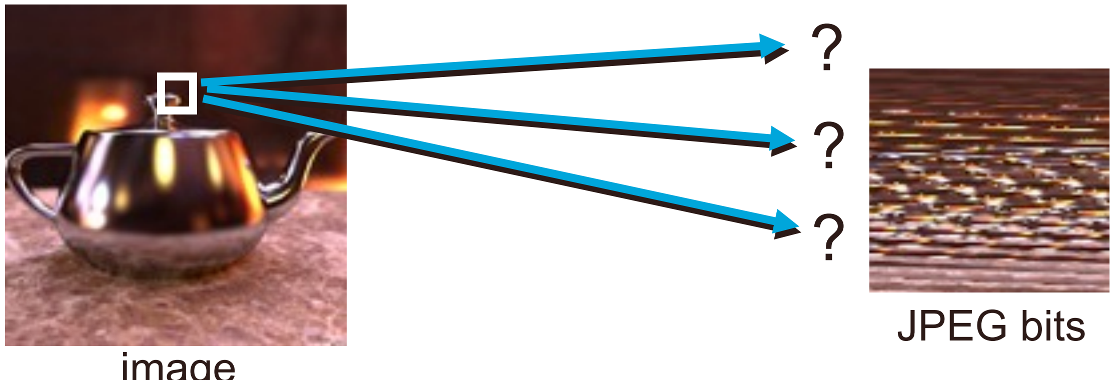
## why not just use JPEG?

- However, variable bit rate also means that you cannot calculate the address for a pixel in the JPEG bits.

# Difference to Image Compression

## why not just use JPEG?

- However, variable bit rate also means that you cannot calculate the address for a pixel in the JPEG bits.

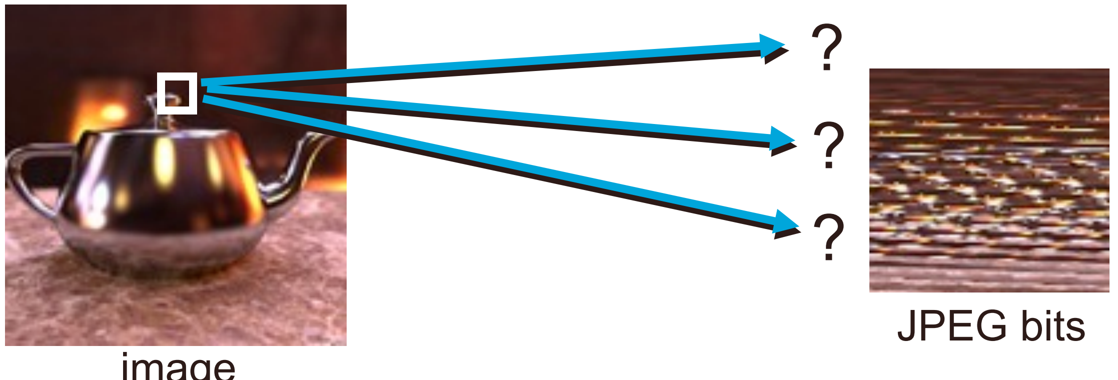



left image courtesy of Henrik Wann Jensen

# Difference to Image Compression

## why not just use JPEG?

- However, variable bit rate also means that you cannot calculate the address for a pixel in the JPEG bits.
- In order to know the address for a particular pixel, you have to parse the entire file.

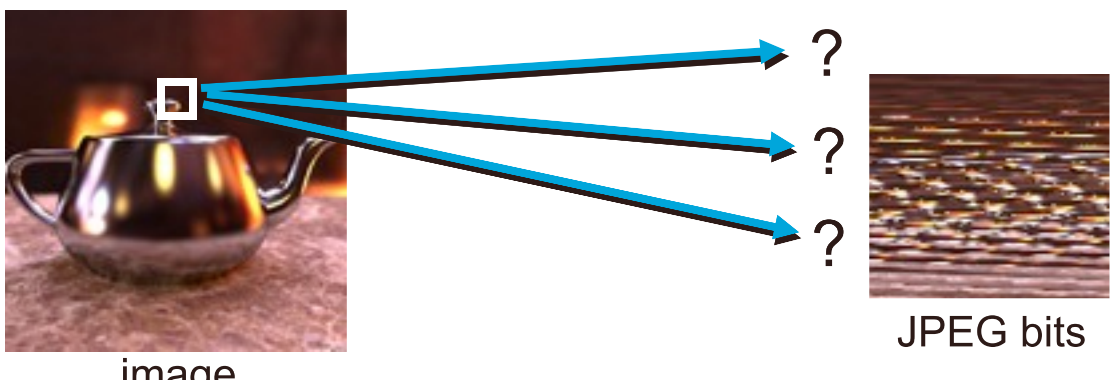



left image courtesy of Henrik Wann Jensen

# Difference to Image Compression

## why not just use JPEG?

- However, variable bit rate also means that you cannot calculate the address for a pixel in the JPEG bits.
- In order to know the address for a particular pixel, you have to parse the entire file.
- During rendering, you would have to parse the entire file for every texel! Not feasible.

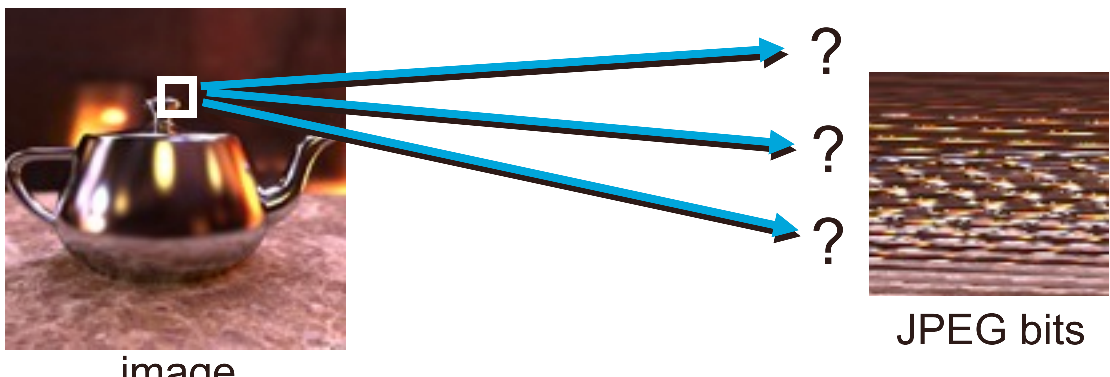



left image courtesy of Henrik Wann Jensen

# Difference to Image Compression

## why not just use JPEG?

- Therefore, most texture compression coders are fixed rate coders, which means that each block in the image occupies the same number of bits, for instance 64 bits per 4x4 block.
- In this way, it is simple to calculate the address for a particular block in the compressed texture bit stream.

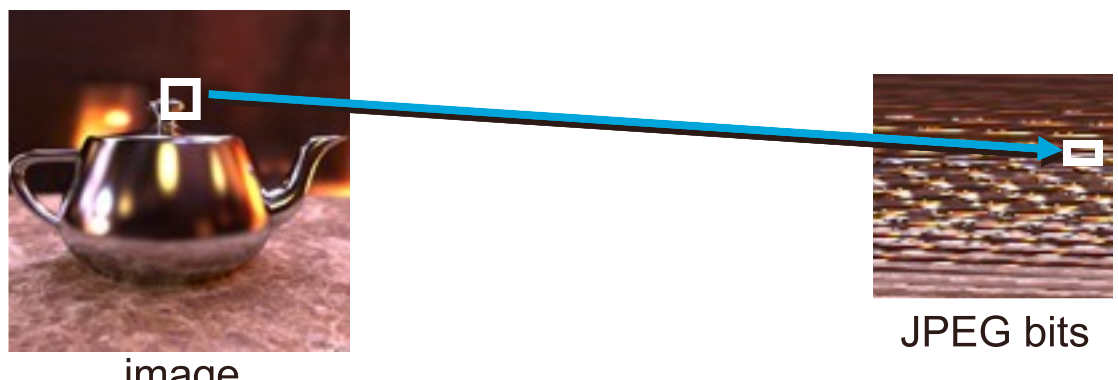



left image courtesy of Henrik Wann Jensen

# Difference to Image Compression

## why not just use JPEG?

- Therefore, most texture compression coders are fixed rate coders, which means that each blocks in the image occupies the same number of bits, for instance 64 bits per 4x4 block.
- In this way, it is simple to calculate the address for a particular block in the compressed texture bit stream.

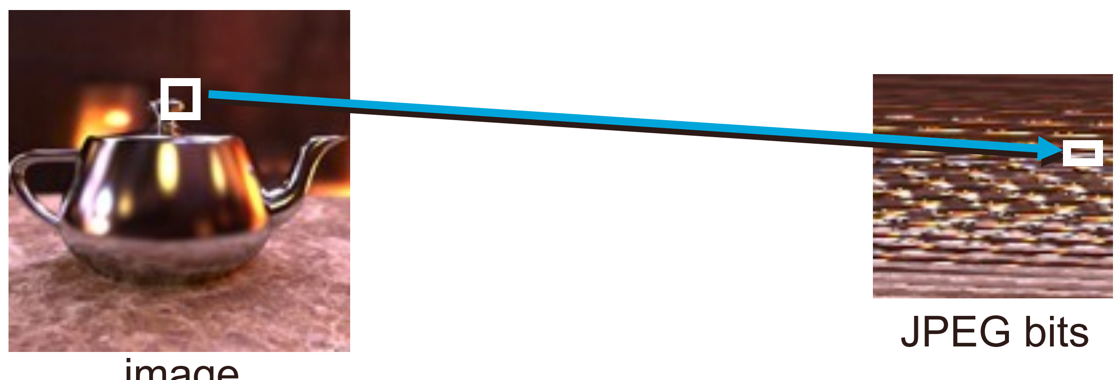



left image courtesy of Henrik Wann Jensen

# Difference to Image Compression

## why not just use JPEG?

- Therefore, most texture compression coders are fixed rate coders, which means that each blocks in the image occupies the same number of bits, for instance 64 bits per 4x4 block.
- In this way, it is simple to calculate the address for a particular block in the compressed texture bit stream.

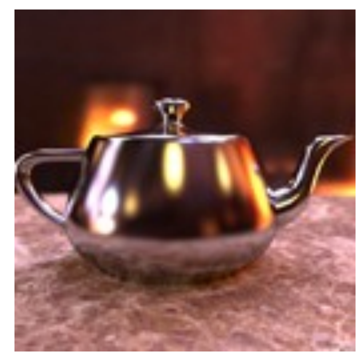



left image courtesy of Henrik Wann Jensen

# Difference to Image Compression

## why not just use JPEG?

- Note, that there is only one rate that will guarantee error free coding, and that is to have no compression at all!
- Thus, for fixed rate coding, one has to allow error (distortion) in the image. The goal is to make this error as small as possible.



original

≈

compressed and decompressed

left image courtesy of Henrik Wann Jensen

© Ericsson AB 2009 | Ericsson Internal | X (X) | Date

ERICSSON

# Difference to Image Compression

## why not just use JPEG?

- Decompression should be of low complexity.
- Up to eight texels must be decompressed for each pixel.
- If we are unlucky, all eight texels can be in different blocks.

texture mipmap levels

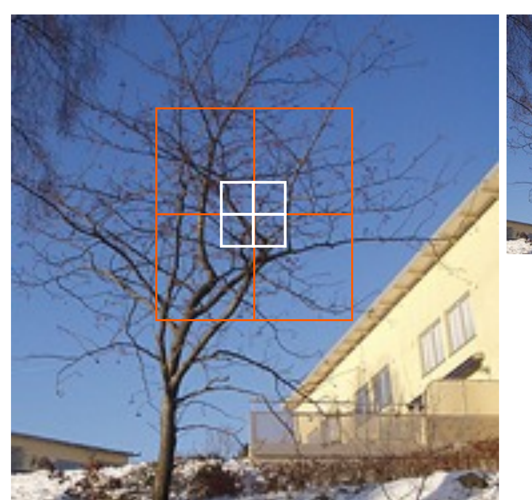
drawn image

# Difference to Image Compression

why not just use JPEG?

- Decompression should be of low complexity.
- Up to eight texels must be decompressed for each pixel.
- If we are unlucky, all eight texels can be in different blocks.

texture mipmap levels



- This means that we have to have eight parallel block decompressors on the chip to deliver one filtered pixel per clock.

# The Texture Cache

- In a system without texture compression, a dedicated texture cache is usually present.

BUS

GPU

Texels

Display

texture cache

# The Texture Cache

› If texture compression is added, the decompression can either happen before or after caching.

BUS

GPU

Texels

Display

texture cache

© Ericsson AB 2009 | Ericsson Internal | X (X) | Date

# The Texture Cache

› If decompression is done before caching, the decompression is allowed to be slower since the data rate out of it is rather low.

BUS

GPU

Texels

Display

Memory

texture decompression

texture cache

# The Texture Cache

- On the other hand, if decompression is done after caching, the cache can be reduced by a factor of, e.g., 6 in terms of surface area.

BUS

GPU

Texels

Display

Memory

texture cache

texture decompression

# The Texture Cache

- On the other hand, if decompression is done after caching, the cache can be reduced by a factor of, e.g., 6 in terms of surface area.
- Complexity should therefore be low enough for handling the larger data streams after caching.

BUS

GPU

Texels

Display

Memory

texture cache

texture decompression

# Palettes and other Global Data

- Many image compression formats have a palette where a number of colors are stored.

index data

| 1 | 12 | 15 | 1 |
|---|---|---|---|
| 1 | 14 | 14 | 1 |
| 1 | 12 | 12 | 1 |
| 1 | 12 | 12 | 1 |

palette

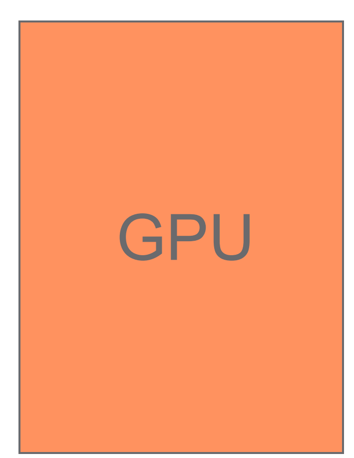
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14 (R=64, G=41, B=29)
15

# Palettes and other Global Data

- This is an indirect way of obtaining the color data.

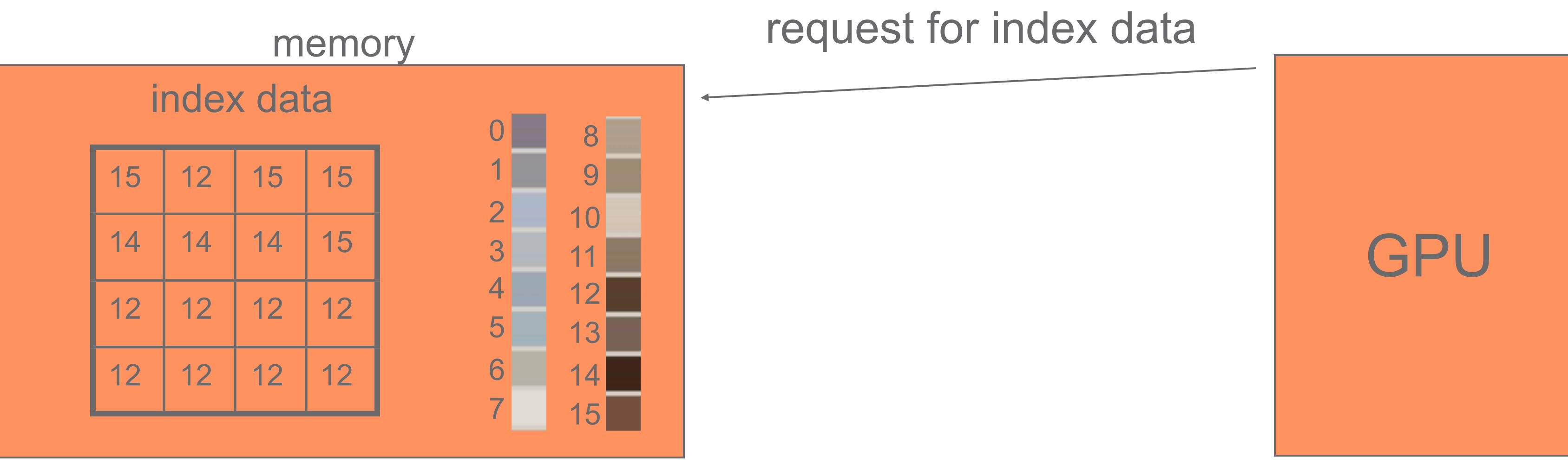
# Palettes and other Global Data

- This is an indirect way of obtaining the color data.



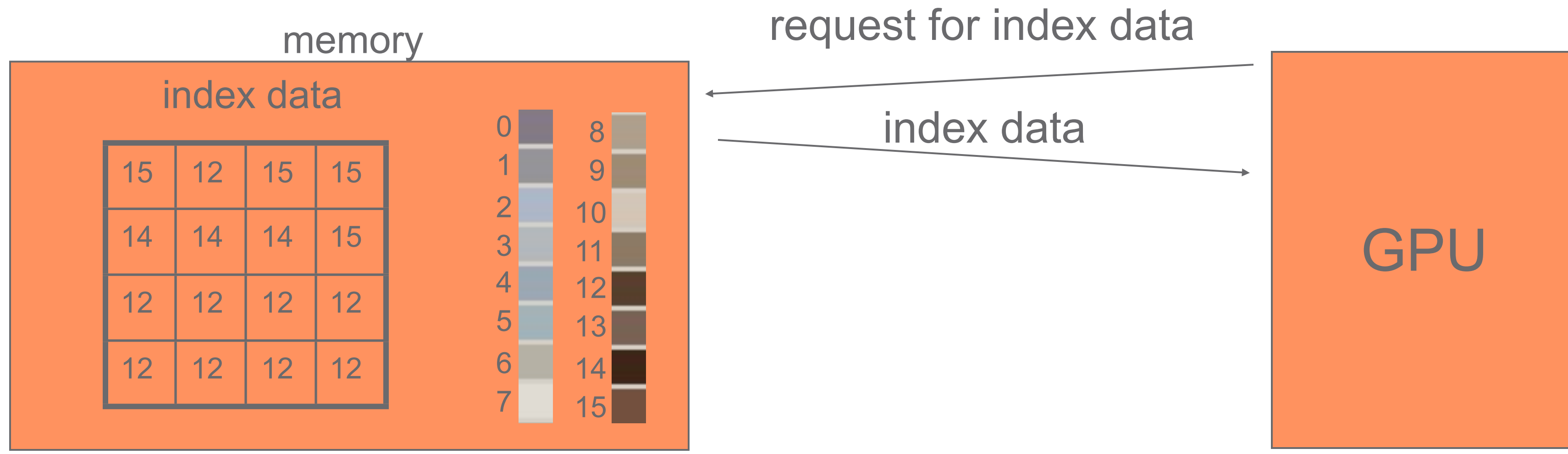
# Palettes and other Global Data

- This is an indirect way of obtaining the color data.
- The GPU must first load the index data



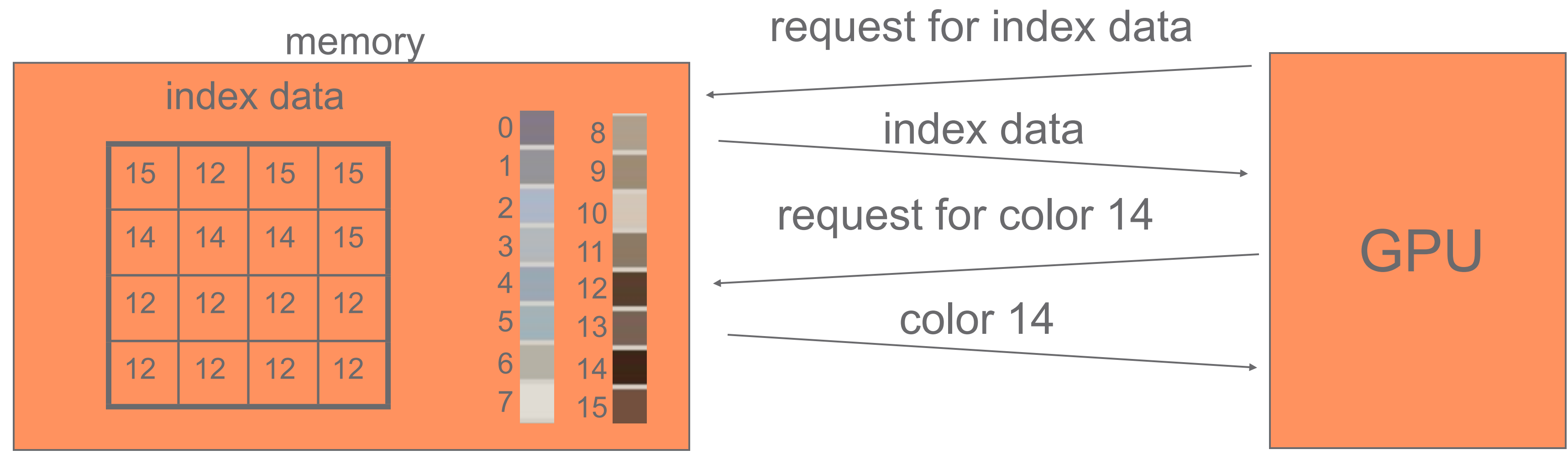
# Palettes and other Global Data

- This is an indirect way of obtaining the color data.
- The GPU must first load the index data



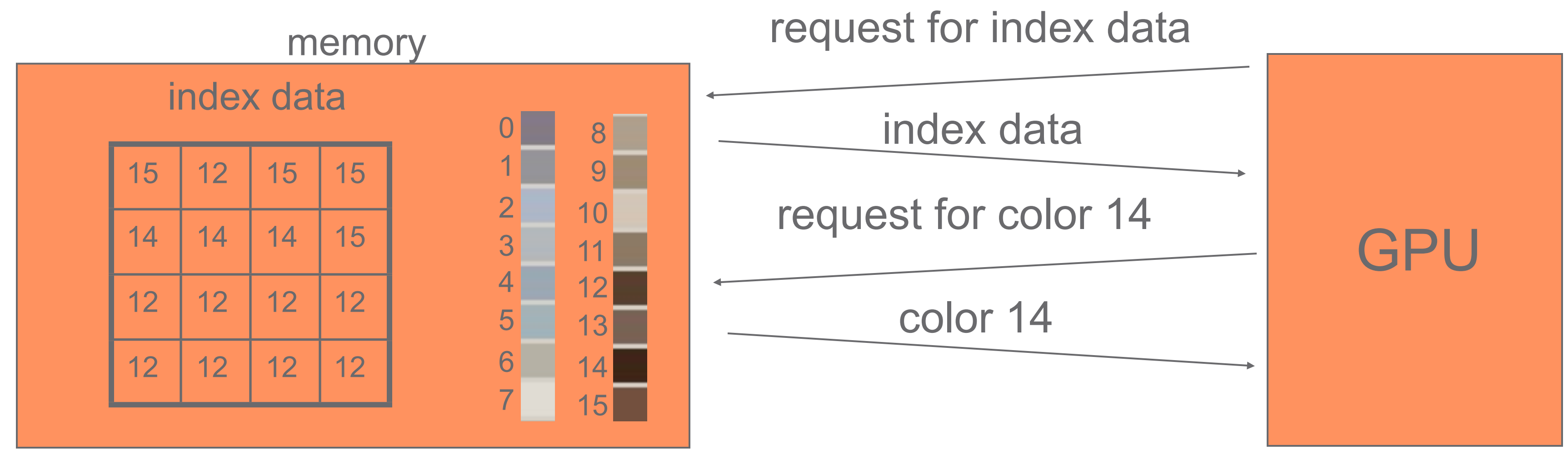
# Palettes and other Global Data

- This is an indirect way of obtaining the color data.
- The GPU must first load the index data
- Only once it has the index data can it load the real color



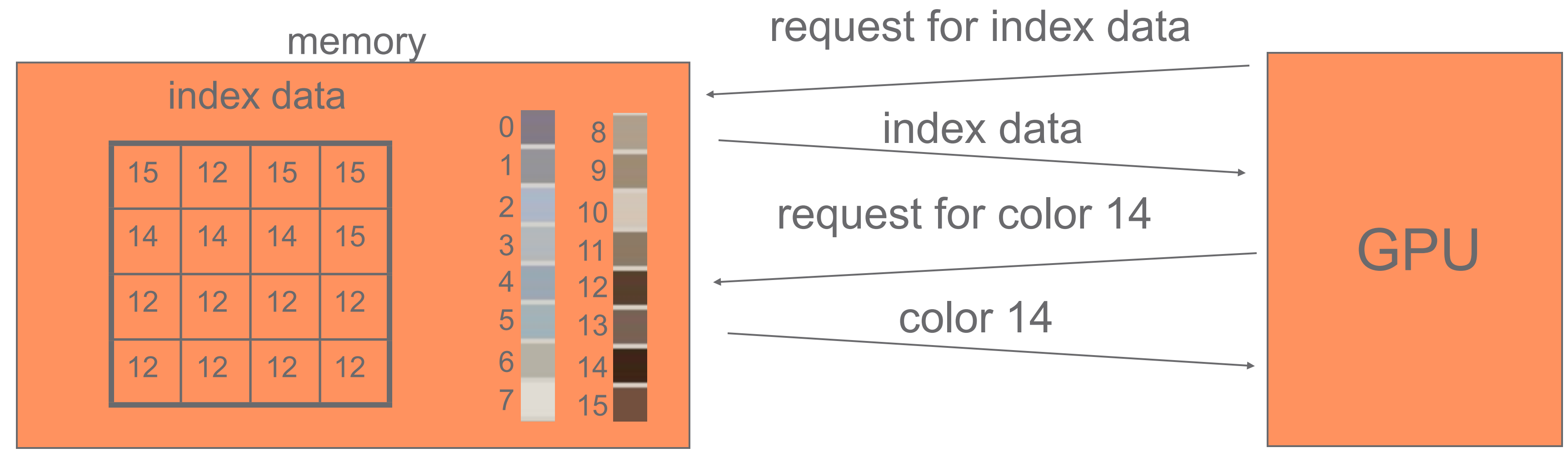
# Palettes and other Global Data

- This is an indirect way of obtaining the color data.
- The GPU must first load the index data
- Only once it has the index data can it load the real color
- This induces extra latency that is costly to hide in FIFO buffers etc.



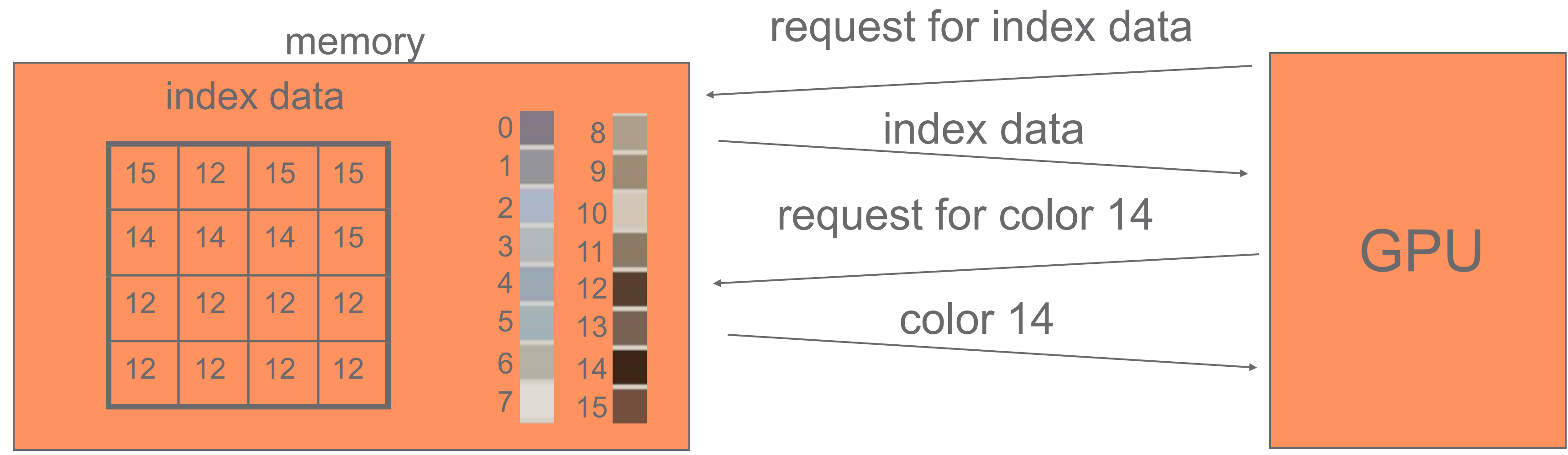
# Palettes and other Global Data

- This is an indirect way of obtaining the color data.
- The GPU must first load the index data
- Only once it has the index data can it load the real color
- This induces extra latency that is costly to hide in FIFO buffers etc.
- Having the table on-chip is expensive, as it can take up as much data as the texture cache itself



# Palettes and other Global Data

- This is an indirect way of obtaining the color data.
- The GPU must first load the index data
- Only once it has the index data can it load the real color
- This induces extra latency that is costly to hide in FIFO buffers etc.
- Having the table on-chip is expensive, as it can take up as much data as the texture cache itself
- Therefore, palettes and other texture depending global data is best avoided.



# Differences to Image Compression

## Summary

1. Random access is needed – fixed rate coder makes this possible.
2. Several parallel units needed – low hardware decompression complexity necessary. (Long compression times OK though!)
3. Indirect addressing due to use of palettes or other global, texture depending data should be avoided.

# Texture Compression Formats

# Palettized Textures

- Were used in the past when memory latency was not the limiting factor
- Is used in software renderers on mobile devices, and is part of JSR 184 and OpenGL ES 1.0.

index data

| 1 | 12 | 15 | 1 |
|---|---|---|---|
| 1 | 14 | 14 | 1 |
| 1 | 12 | 12 | 1 |
| 1 | 12 | 12 | 1 |

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14 (R=64, G=41, B=29)
15

# Block Truncation Coding

# BTC – Block Truncation Coding

- Image is divided into 4x4 blocks
- Two 8-bit gray shades are encoded per block

col 0

col 1

# BTC – Block Truncation Coding

- Image is divided into 4x4 blocks
- Two 8-bit gray shades are encoded per block
- A bit mask of 16 bits is also used.

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

# BTC – Block Truncation Coding

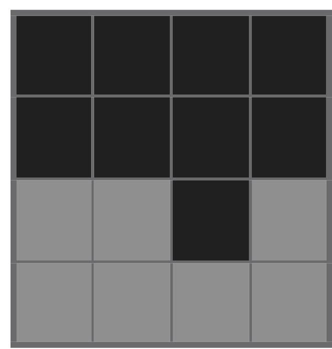
- Image is divided into 4x4 blocks
- Two 8-bit gray shades are encoded per block
- A bit mask of 16 bits is also used.

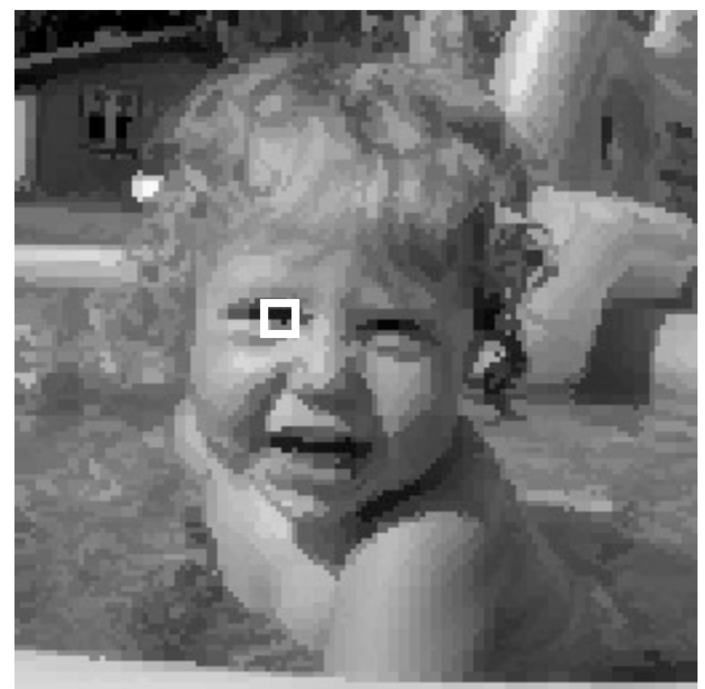
| 0 | 0 | 0 | 0 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

# BTC – Block Truncation Coding

- Image is divided into 4x4 blocks
- Two 8-bit gray shades are encoded per block
- A bit mask of 16 bits is also used.

| 0 | 0 | 0 | 0 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |



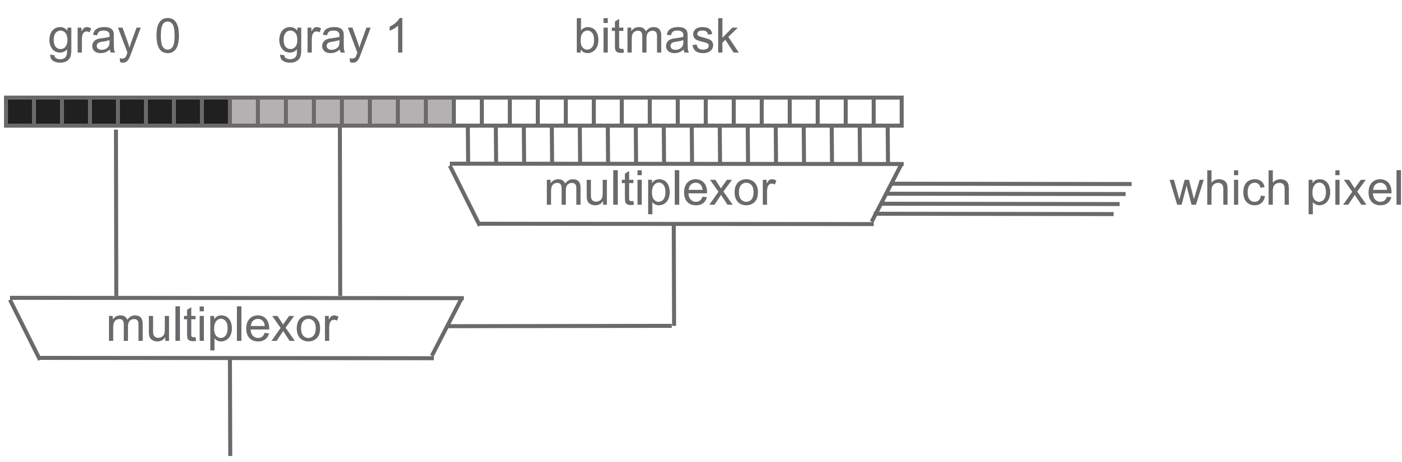


# BTC – Block Truncation Coding

- Bit rate equals 8+8+16 = 32 bits per block, i.e., 2 bits per pixel (bpp).
- Everything is contained in the codeword, no “global data” or color palette needs to be read.

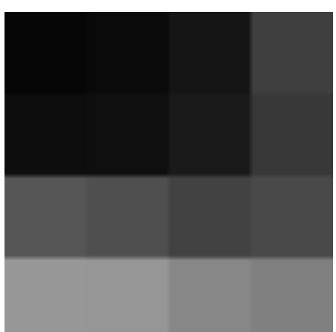
# BTC – Block Truncation Coding

- Bit rate equals 8+8+16 = 32 bits per block, i.e., 2 bits per pixel (bpp).
- Everything is contained in the codeword, no “global data” or color palette needs to be read.
- Hardware complexity for decompression is very simple:



# BTC – Block Truncation Coding
## compression

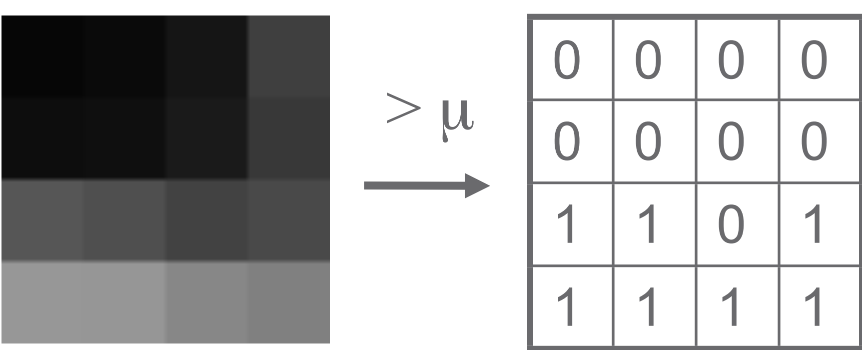
- First the mean $\mu$ and standard deviation $s$ of the block is calculated.



# BTC – Block Truncation Coding

## Compression

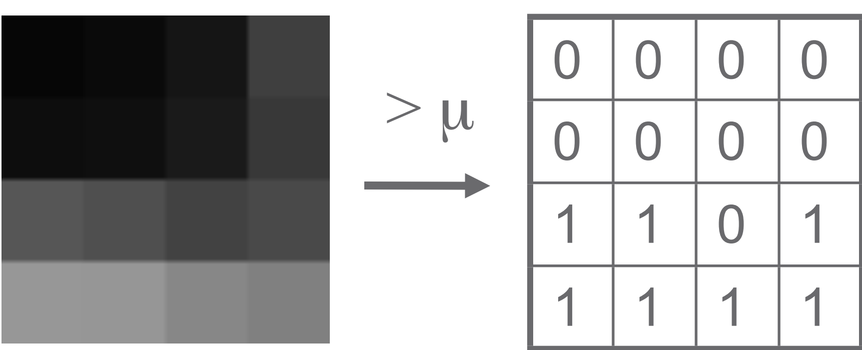
- First the mean $\mu$ and standard deviation $s$ of the block is calculated.
- Then the bit mask is constructed. All pixels with gray value greater than $\mu$ equals $1$, otherwise $0$.



# BTC – Block Truncation Coding

## Compression

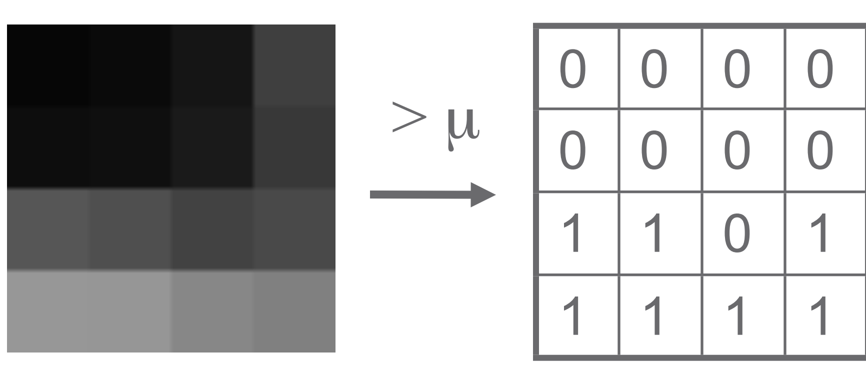
- First the mean $\mu$ and standard deviation $s$ of the block is calculated.
- Then the bit mask is constructed. All pixels with gray value greater than $\mu$ equals $1$, otherwise $0$.
- Let $q$ be the number bigger than $\mu$ (7 in our case), and $m$ be the total number of pixels.



# BTC – Block Truncation Coding

## Compression

- First the mean $\mu$ and standard deviation $s$ of the block is calculated.
- Then the bit mask is constructed. All pixels with gray value greater than $\mu$ equals $1$, otherwise $0$.
- Let $q$ be the number bigger than $\mu$ (7 in our case), and $m$ be the total number of pixels.




- The colors can now be calculated as

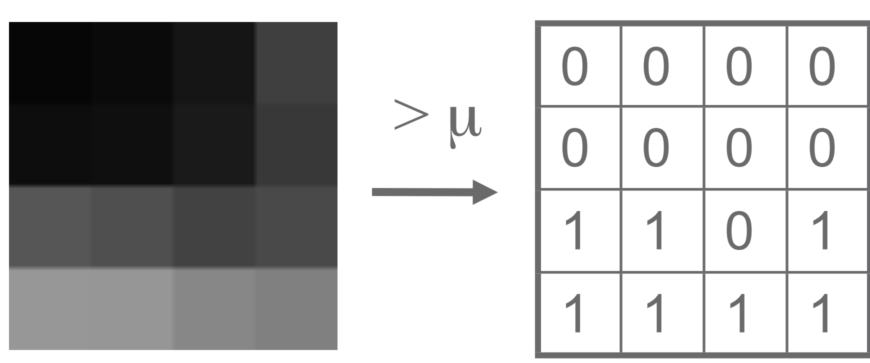
$$\text{col0} = \mu - s\sqrt{\frac{q}{m\text{-}q}}$$

$$\text{col1} = \mu + s\sqrt{\frac{m\text{-}q}{q}}$$

# BTC – Block Truncation Coding

## Compression

- First the mean $\mu$ and standard deviation $s$ of the block is calculated.
- Then the bit mask is constructed. All pixels with gray value greater than $\mu$ equals $1$, otherwise $0$.
- Let $q$ be the number bigger than $\mu$ (7 in our case), and $m$ be the total number of pixels.




| 0 | 0 | 0 | 0 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

- The colors can now be calculated as

$$\text{col0} = \mu - s\sqrt{\frac{q}{m-q}}$$

$$\text{col1} = \mu + s\sqrt{\frac{m-q}{q}}$$

- Or, just do exhaustive search!

# BTC – Block Truncation Coding

## Quality

- This means that the mean and the standard deviation of the block is preserved.
- However, having only two shades of gray gives rise to banding artifacts.

# BTC – Block Truncation Coding

## Quality

- This means that the mean and the standard deviation of the block is preserved.
- However, having only two shades of gray gives rise to banding artifacts.

original

BTC

# BTC – Block Truncation Coding

## Quality

original

BTC

# BTC – Block Truncation Coding

## Color Compression

- BTC can also be used separately on the Red, Green and Blue Components. Bit rate then becomes 6 bpp.
- Still, banding artifacts remain, and shot noise of strangely colored pixels appear.

# BTC – Block Truncation Coding

## Color Compression

- BTC can also be used separately on the Red, Green and Blue Components. Bit rate then becomes 6 bpp.
- Still, banding artifacts remain, and shot noise of strangely colored pixels appear.

original

BTC

# BTC – Block Truncation Coding

## Color Compression

- BTC can also be used separately on the Red, Green and Blue Components. Bit rate then becomes 6 bpp.
- Still, banding artifacts remain, and shot noise of strangely colored pixels appear.

original

BTC

© Ericsson AB 2009 | Ericsson Internal | X (X) | Date

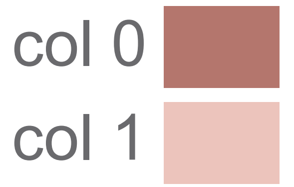
# Color Cell Compression

# CCC – Color Cell Compression

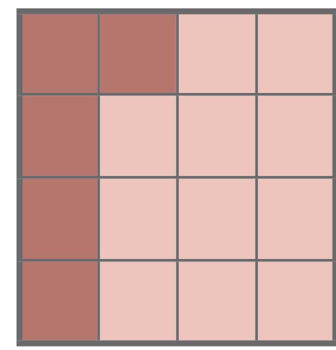
- Based on BTC, but instead of two gray scales, two colors are used per block, in RGB565 format.

# CCC – Color Cell Compression

- Based on BTC, but instead of two gray scales, two colors are used per block, in RGB565 format.




| 0 | 0 | 1 | 1 |
|---|---|---|---|
| 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 |

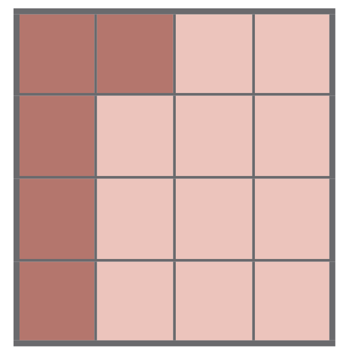


(fake)

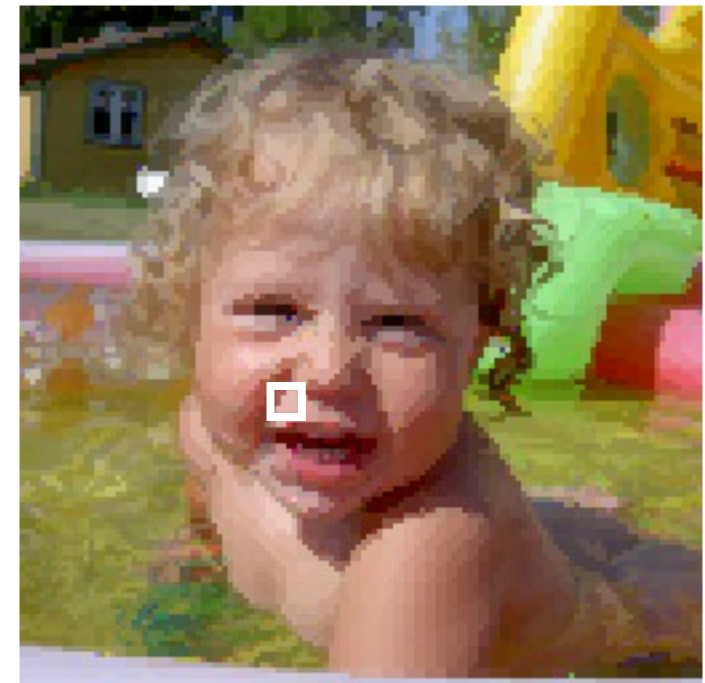
# CCC – Color Cell Compression

- Based on BTC, but instead of two gray scales, two colors are used per block, in RGB565 format.
- Two 16 bit colors, together with the 16-bit-wide bit mask, gives 48 bits per block or 3 bpp.

| | | | |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 |



(fake)



# CCC – Color Cell Compression

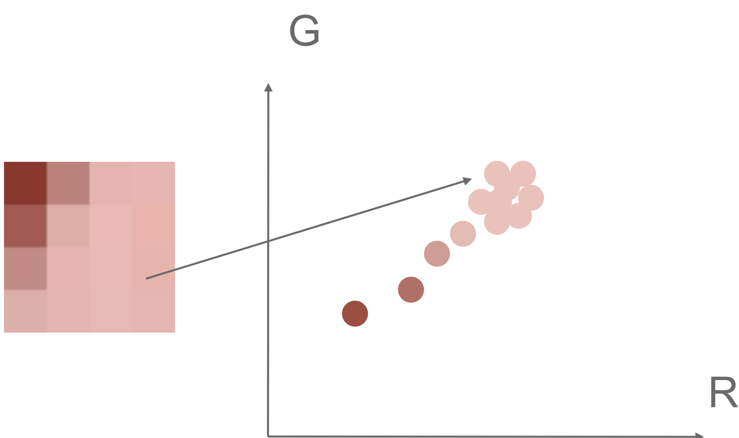
## Compression

- To compress a block, the LBG-algorithm can be used.
- Plot colors in block as points in RGB space

# CCC – Color Cell Compression

## Compression

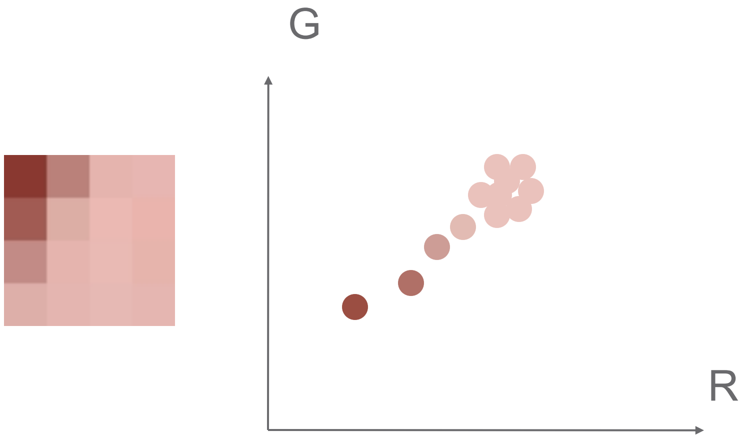
- To compress a block, the LGB-algorithm can be used.
- Plot colors in block as points in RGB space



# CCC – Color Cell Compression

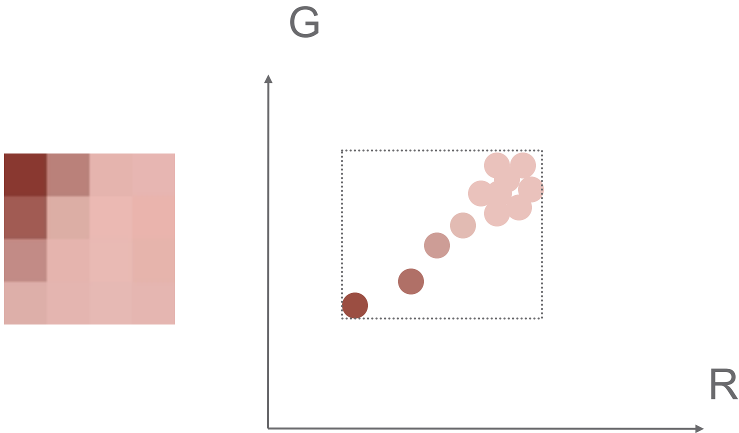
## Compression

- To compress a block, the LGB-algorithm can be used.
- Plot colors in block as points in RGB space
- Start with two random colors in the bounding box



# CCC – Color Cell Compression
## Compression

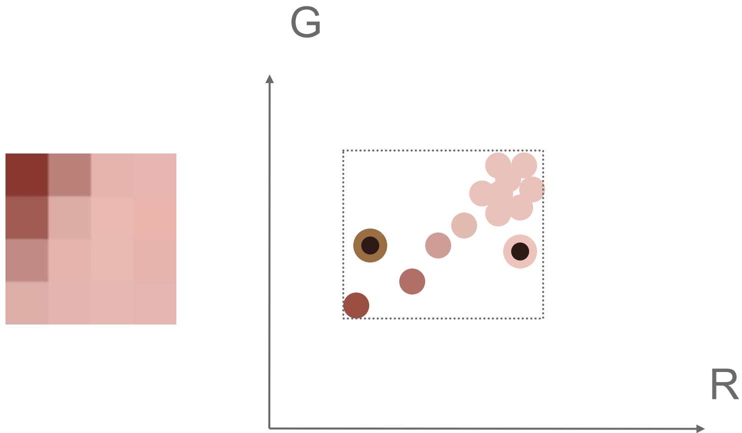
- To compress a block, the LGB-algorithm can be used.
- Plot colors in block as points in RGB space
- Start with two random colors in the bounding box



# CCC – Color Cell Compression

## Compression

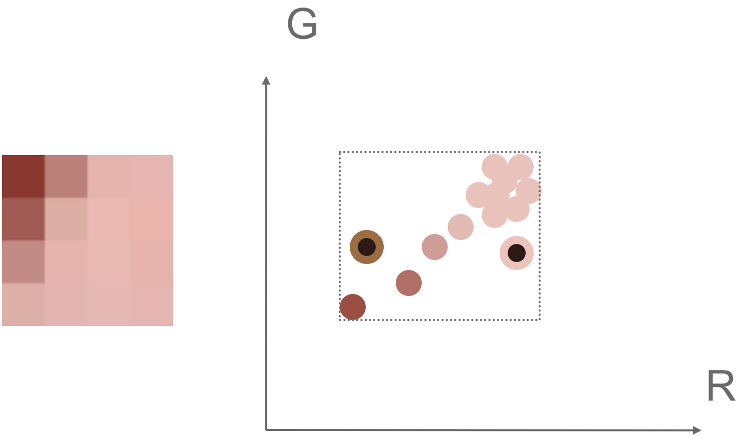
- To compress a block, the LGB-algorithm can be used.
- Plot colors in block as points in RGB space
- Start with two random colors in the bounding box



# CCC – Color Cell Compression

## Compression

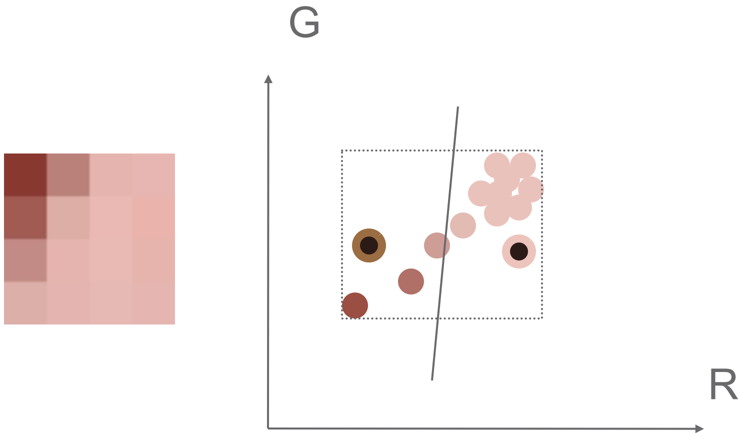
- To compress a block, the LGB-algorithm can be used.
- Plot colors in block as points in RGB space
- Start with two random colors in the bounding box
- See what color each point is closest to



# CCC – Color Cell Compression

## Compression

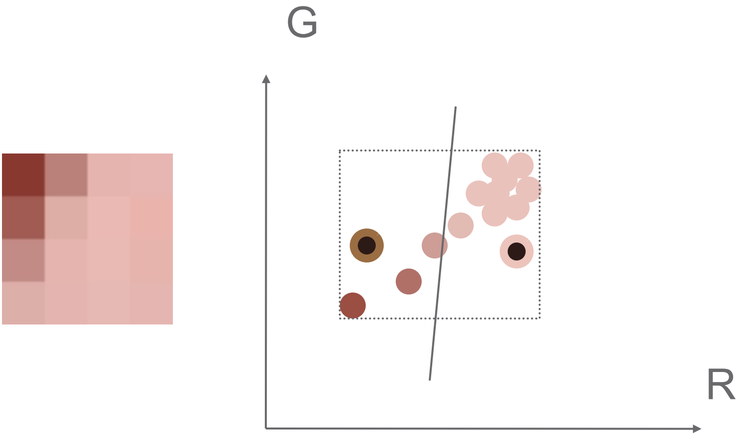
- To compress a block, the LGB-algorithm can be used.
- Plot colors in block as points in RGB space
- Start with two random colors in the bounding box
- See what color each point is closest to



# CCC – Color Cell Compression

## Compression

- To compress a block, the LGB-algorithm can be used.
- Plot colors in block as points in RGB space
- Start with two random colors in the bounding box
- See what color each point is closest to
- Refine the colors to the average of its points

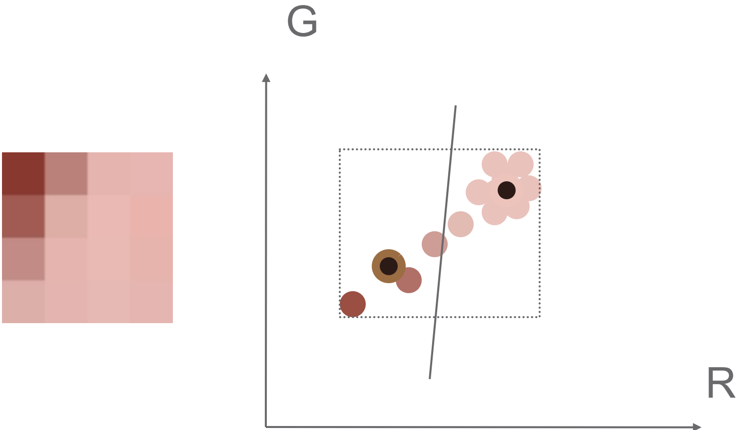



© Ericsson AB 2009 | Ericsson Internal | X (X) | Date

# CCC – Color Cell Compression

## Compression

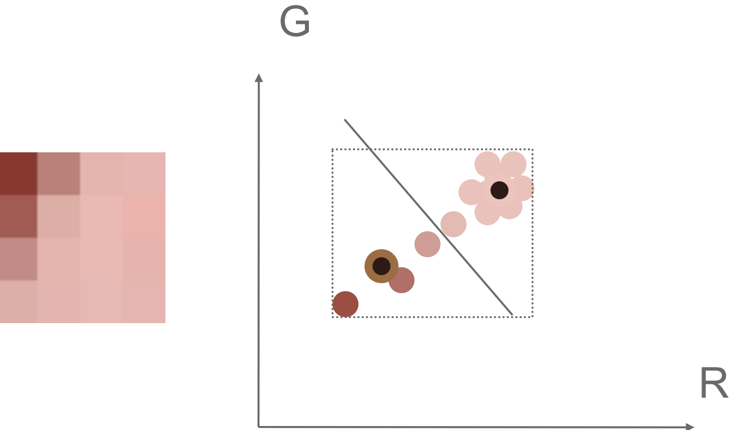
- To compress a block, the LGB-algorithm can be used.
- Plot colors in block as points in RGB space
- Start with two random colors in the bounding box
- See what color each point is closest to
- Refine the colors to the average of its points



# CCC – Color Cell Compression

## Compression

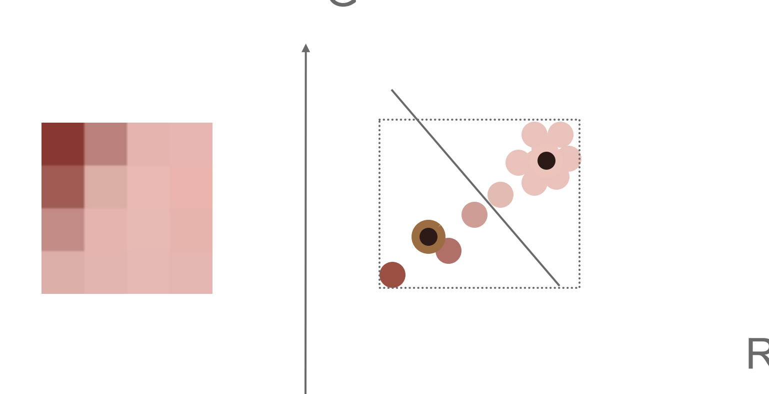
- To compress a block, the LGB-algorithm can be used.
- Plot colors in block as points in RGB space
- Start with two random colors in the bounding box
- See what color each point is closest to
- Refine the colors to the average of its points
- See again what color each point is closest to, etc.



# CCC – Color Cell Compression

## Compression

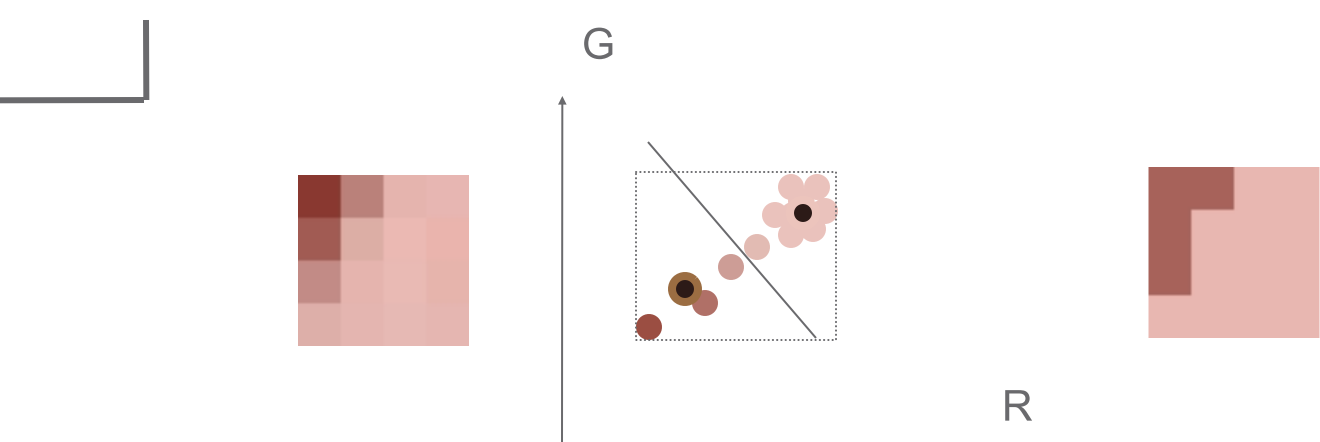
- To compress a block, the LGB-algorithm can be used.
- Plot colors in block as points in RGB space
- Start with two random colors in the bounding box
- See what color each point is closest to
- Refine the colors to the average of its points
- See again what color each point is closest to, etc.



# CCC – Color Cell Compression

## Compression

- To compress a block, the LGB-algorithm can be used.
- Plot colors in block as points in RGB space
- Start with two random colors in the bounding box
- See what color each point is closest to
- Refine the colors to the average of its points
- See again what color each point is closest to, etc.



# CCC – Color Cell Compression

## 2-bit version

- Campbell et al. also present a 2-bit version of CCC. Here, the 16-bit RGB565 colors are changed to 8-bit indexes into a 256 wide color palette.
- However, this introduces latency as discussed above.
- Quality in both 3- and 2-bit versions is not too great, since only two colors per 4x4 block is possible.
  - Block artifacts visible (“I can see the blocks”)
  - Banding artifacts (“Smooth transitions comes in steps.”)

# S3TC

# S3TC – S3 Texture Compression

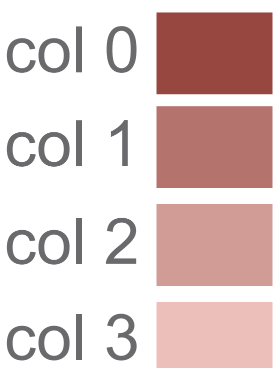
## also called DXT1

- S3TC can be seen as an extension of CCC.
- Instead of two colors, four colors can be chosen per pixel.

col 3

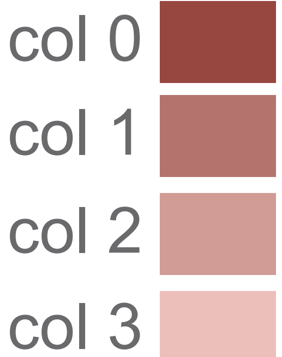
# S3TC – S3 Texture Compression
## also called DXT1

- S3TC can be seen as an extension of CCC.
- Instead of two colors, four colors can be chosen per pixel.



# S3TC – S3 Texture Compression
## also called DXT1

- S3TC can be seen as an extension of CCC.
- Instead of two colors, four colors can be chosen per pixel. However, only col 0 and col 3 are stored in the block. Col 1 and col 2 are linearly interpolated

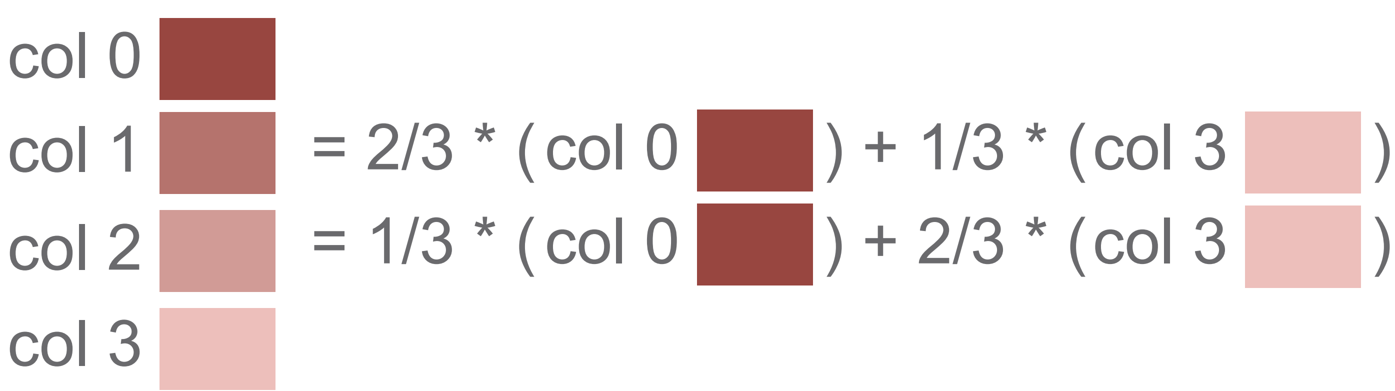


# S3TC – S3 Texture Compression

also called DXT1

› S3TC can be seen as an extension of CCC.

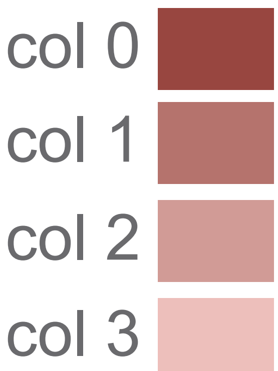
› Instead of two colors, four colors can be chosen per pixel. However, only col 0 and col 3 are stored in the block. Col 1 and col 2 are linearly interpolated



# S3TC – S3 Texture Compression

also called DXT1

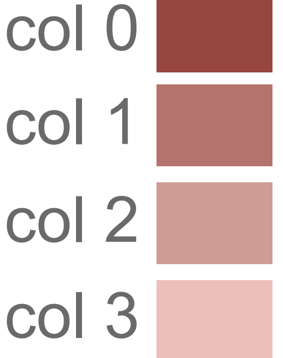
- S3TC can be seen as an extension of CCC.
- Instead of two colors, four colors can be chosen per pixel. However, only col 0 and col 3 are stored in the block. Col 1 and col 2 are linearly interpolated
- Bit mask must now be two bits per pixel



# S3TC – S3 Texture Compression

also called DXT1

- S3TC can be seen as an extension of CCC.
- Instead of two colors, four colors can be chosen per pixel. However, only col 0 and col 3 are stored in the block. Col 1 and col 2 are linearly interpolated
- Bit mask must now be two bits per pixel

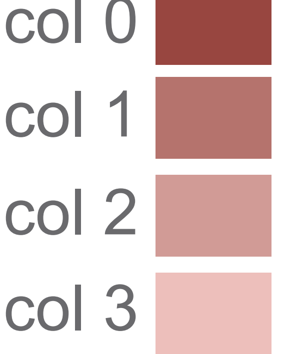



| | | | |
|---|---|---|---|
| 00 | 10 | 11 | 11 |
| 01 | 11 | 11 | 11 |
| 10 | 11 | 11 | 11 |
| 11 | 11 | 11 | 11 |

# S3TC – S3 Texture Compression

also called DXT1

- S3TC can be seen as an extension of CCC.
- Instead of two colors, four colors can be chosen per pixel. However, only col 0 and col 3 are stored in the block. Col 1 and col 2 are linearly interpolated
- Bit mask must now be two bits per pixel



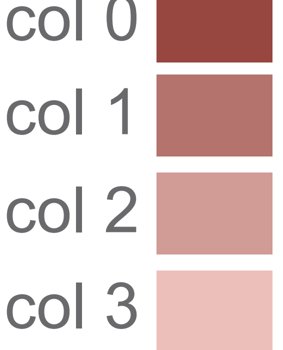

| | | | |
|---|---|---|---|
| 00 | 10 | 11 | 11 |
| 01 | 11 | 11 | 11 |
| 10 | 11 | 11 | 11 |
| 11 | 11 | 11 | 11 |

| | | | |
|---|---|---|---|
| 0 | 2 | 3 | 3 |
| 1 | 3 | 3 | 3 |
| 2 | 3 | 3 | 3 |
| 3 | 3 | 3 | 3 |

# S3TC – S3 Texture Compression

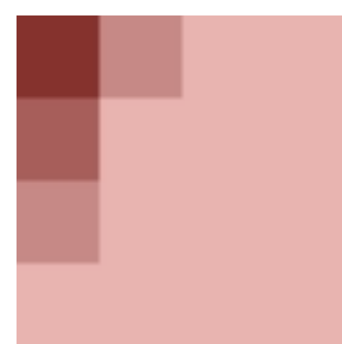
also called DXT1

- S3TC can be seen as an extension of CCC.
- Instead of two colors, four colors can be chosen per pixel. However, only col 0 and col 3 are stored in the block. Col 1 and col 2 are linearly interpolated
- Bit mask must now be two bits per pixel




| 00 | 10 | 11 | 11 |
|---|---|---|---|
| 01 | 11 | 11 | 11 |
| 10 | 11 | 11 | 11 |
| 11 | 11 | 11 | 11 |

| 0 | 2 | 3 | 3 |
|---|---|---|---|
| 1 | 3 | 3 | 3 |
| 2 | 3 | 3 | 3 |
| 3 | 3 | 3 | 3 |



# S3TC – S3 Texture Compression

## quality

- In this way, four colors per 4x4 block can be used instead of two – quality increases tremendously.

# S3TC – S3 Texture Compression

quality

- In this way, four colors per 4x4 block can be used instead of two – quality increases tremendously.

original

S3TC

# S3TC – S3 Texture Compression
quality

- In this way, four colors per 4x4 block can be used instead of two – quality increases tremendously.
- S3TC was adopted by Direct 3D under the name DXT1 and is now the industry standard in the desktop space.

original

S3TC

# S3TC – S3 Texture Compression

quality

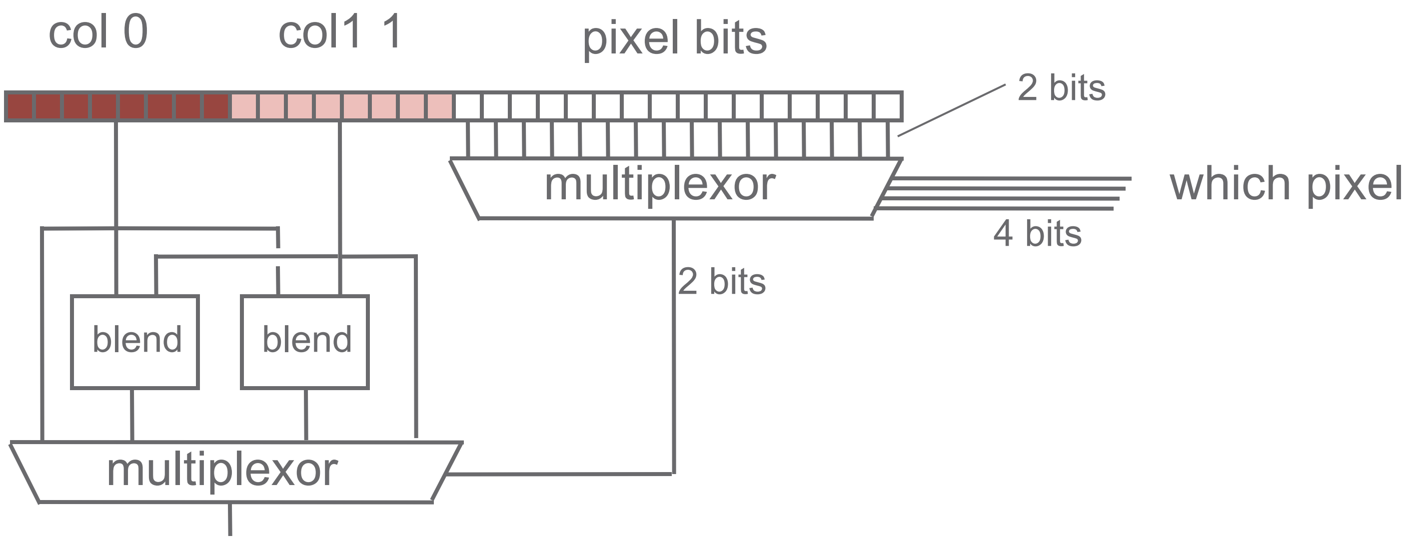
- In this way, four colors per 4x4 block can be used instead of two – quality increases tremendously.
- S3TC was adopted by Direct 3D under the name DXT1 and is now the industry standard in the desktop space.

# S3TC – S3 Texture Compression

- The two base colors are stored in RGB565 (16 bits). Together with the 32 bits of pixel bits we get 64 bits per block, or 4 bpp. Compression ratio is thus 6:1.

# S3TC – S3 Texture Compression

- The two base colors are stored in RGB565 (16 bits). Together with the 32 bits of pixel bits we get 64 bits per block, or 4 bpp. Compression ratio is thus 6:1.
- Decompression includes multiplication of 1/3 and 2/3.

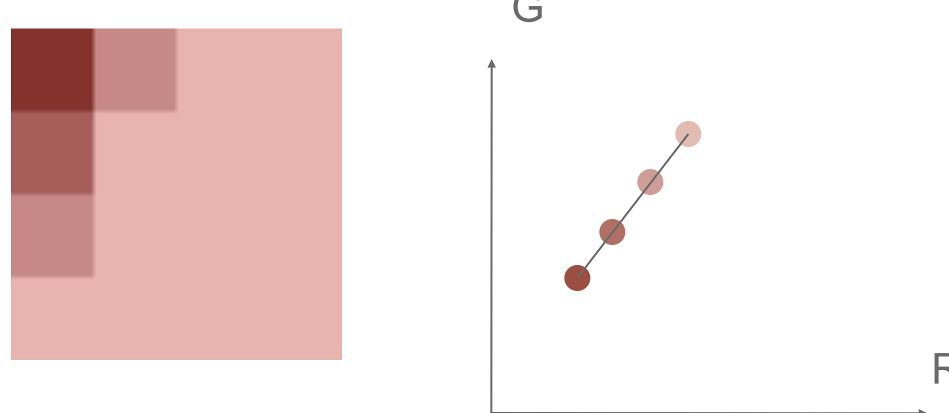


# S3TC – S3 Texture Compression

- Due to the way the intermediate colors are interpolated, the four colors of the block will lie on a straight line in RGB space.

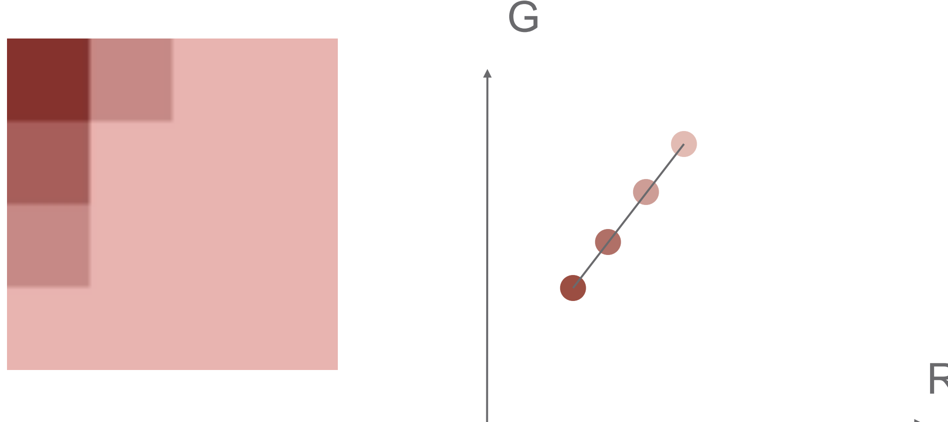
# S3TC – S3 Texture Compression

- Due to the way the intermediate colors are interpolated, the four colors of the block will lie on a straight line in RGB space.



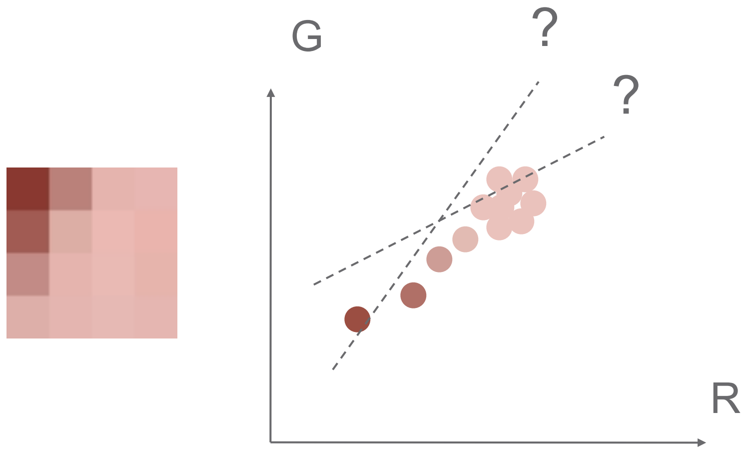
# S3TC – S3 Texture Compression

- Due to the way the intermediate colors are interpolated, the four colors of the block will lie on a straight line in RGB space.
- For many natural images, this is a rather good approximation.



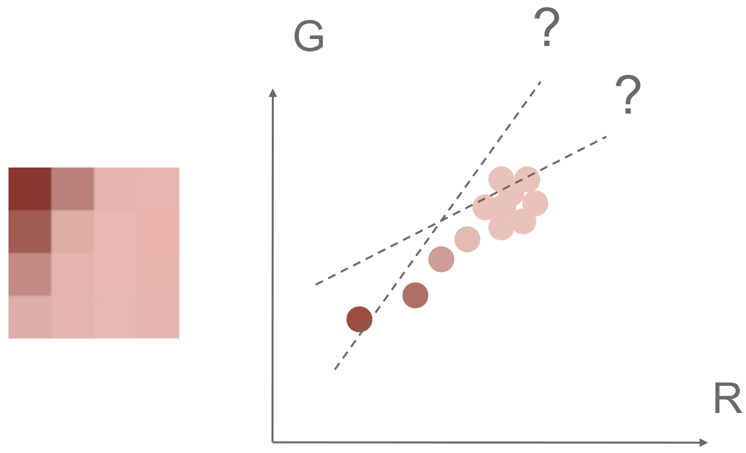
# S3TC – S3 Texture Compression

- One way to compress blocks to S3TC is to look for this line, or “major axis” in the data.



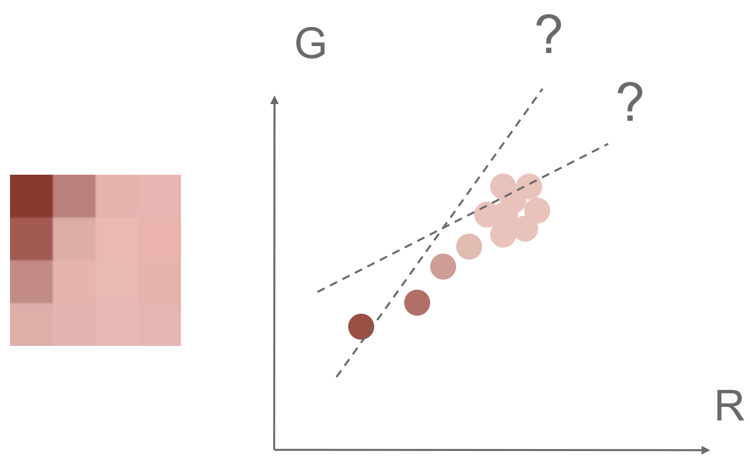
# S3TC – S3 Texture Compression

- One way to compress blocks to S3TC is to look for this line, or “major axis” in the data.
- A tool from statistics, Principal Component Analysis (PCA) can be used to find the line.



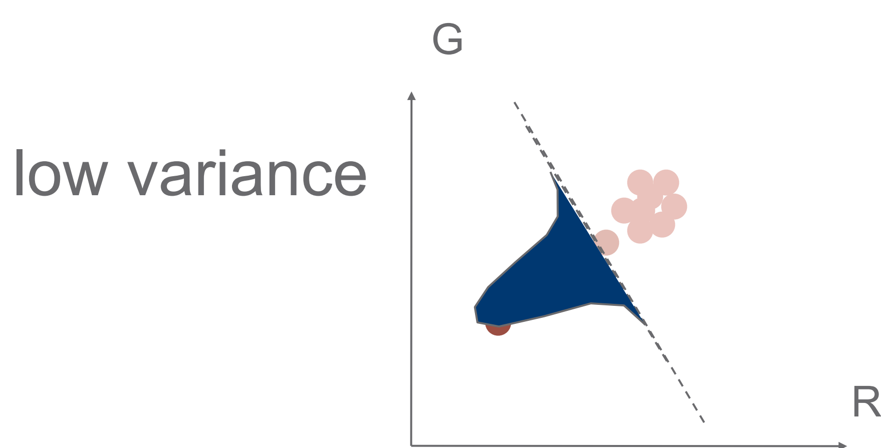
# S3TC – S3 Texture Compression

- One way to compress blocks to S3TC is to look for this line, or “major axis” in the data.
- A tool from statistics, Principal Component Analysis (PCA) can be used to find the line.
- PCA finds the direction, along which the points should be projected, so that they have maximal variance.



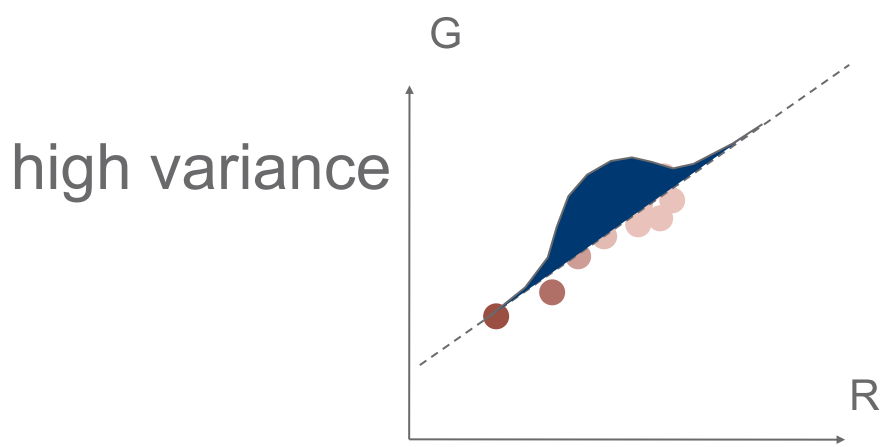
# S3TC – S3 Texture Compression

- One way to compress blocks to S3TC is to look for this line, or “major axis” in the data.
- A tool from statistics, Principal Component Analysis (PCA) can be used to find the line.
- PCA finds the direction, along which the points should be projected, so that they have maximal variance.



# S3TC – S3 Texture Compression

- One way to compress blocks to S3TC is to look for this line, or “major axis” in the data.
- A tool from statistics, Principal Component Analysis (PCA) can be used to find the line.
- PCA finds the direction, along which the points should be projected, so that they have maximal variance.



# Principal Component Analysis

› First calculate and remove the average from the colors:

# Principal Component Analysis

- First calculate and remove the average from the colors:

$$\mu_r = \frac{1}{16}\sum r_i \,,$$

$$r'_i = r_i - \mu_r \,,$$

# Principal Component Analysis

- First calculate and remove the average from the colors:

$$\mu_r = \frac{1}{16}\sum r_i, \qquad \mu_g = \frac{1}{16}\sum g_i, \qquad \mu_b = \frac{1}{16}\sum b_i,$$

$$r'_i = r_i - \mu_r, \qquad g'_i = g_i - \mu_g, \qquad b'_i = b_i - \mu_b$$

# Principal Component Analysis

- First calculate and remove the average from the colors:

$$\mu_r = \frac{1}{16}\sum r_i\,, \qquad \mu_g = \frac{1}{16}\sum g_i\,, \qquad \mu_b = \frac{1}{16}\sum b_i\,,$$

$$r'_i = r_i - \mu_r\,, \qquad g'_i = g_i - \mu_g\,, \qquad b'_i = b_i - \mu_b$$

- Then, regard the average-free colors of the block as outcomes $x_1$ from a random vector $\mathbf{X}$:

# Principal Component Analysis

- First calculate and remove the average from the colors:

$$\mu_r = \frac{1}{16}\sum r_i , \qquad \mu_g = \frac{1}{16}\sum g_i , \qquad \mu_b = \frac{1}{16}\sum b_i ,$$

$$r'_i = r_i - \mu_r , \qquad g'_i = g_i - \mu_g , \qquad b'_i = b_i - \mu_b$$

- Then, regard the average-free colors of the block as outcomes $x_1$ from a random vector $\mathbf{X}$:
- $x_1 = (r'_1, g'_1, b'_1), x_2 = (r'_2, g'_2, b'_2), \ldots, x_{16} = (r'_{16}, g'_{16}, b'_{16})$ *from* $\mathbf{X}$.

# Principal Component Analysis

- First calculate and remove the average from the colors:

$$\mu_r = \frac{1}{16}\sum r_i, \qquad \mu_g = \frac{1}{16}\sum g_i, \qquad \mu_b = \frac{1}{16}\sum b_i,$$

$$r'_i = r_i - \mu_r, \qquad g'_i = g_i - \mu_g, \qquad b'_i = b_i - \mu_b$$

- Then, regard the average-free colors of the block as outcomes $x_1$ from a random vector $\mathbf{X}$:
- $x_1 = (r'_1, g'_1, b'_1), x_2 = (r'_2, g'_2, b'_2), \ldots, x_{16} = (r'_{16}, g'_{16}, b'_{16})$ *from* $\mathbf{X}$.
- The axis we are interested in is the first eigenvector of the covariance matrix $C_{\mathbf{X}}$ of $\mathbf{X}$.

# Principal Component Analysis
cont.

- The covariance matrix $C_{\mathbf{X}}$ of $\mathbf{X}$ can be estimated using
- $C_{\mathbf{X}} \sim \frac{1}{15} AA^{T},$

$$where\ A = [\,x_1\, x_2 \ldots x_{16}] = \begin{bmatrix} r'_1 & r'_2 & & r'_{16} \\ g'_1 & g'_2 & \cdots & g'_{16} \\ b'_1 & b'_2 & & b'_{16} \end{bmatrix}$$

# Principal Component Analysis

cont.

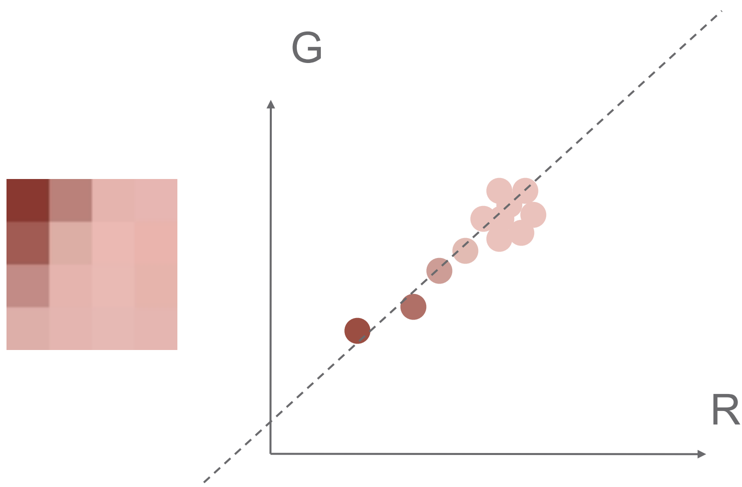
- The covariance matrix $C_{\mathbf{X}}$ of $\mathbf{X}$ can be estimated using
- $C_{\mathbf{X}} \sim \frac{1}{15} AA^{T},$

$$where\ A = [\, x_1\, x_2\, \ldots\, x_{16}] = \begin{bmatrix} r'_1 & r'_2 & & r'_{16} \\ g'_1 & g'_2 & \cdots & g'_{16} \\ b'_1 & b'_2 & & b'_{16} \end{bmatrix}$$

- The major axis is the first eigenvector of $C_{\mathbf{X}}$. The scaling of 1/15 does not change the eigenvector, and $AA^T$ can be used directly.

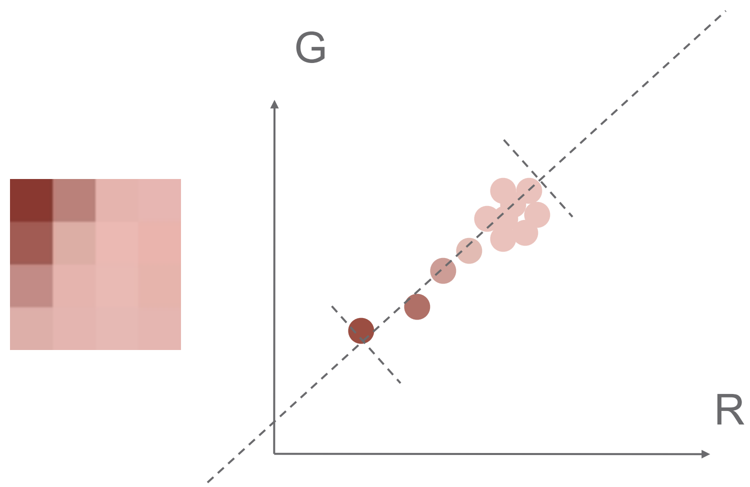
# S3TC – S3 Texture Compression

- Once we have the major axis, it is simply an issue of placing the two outer colors



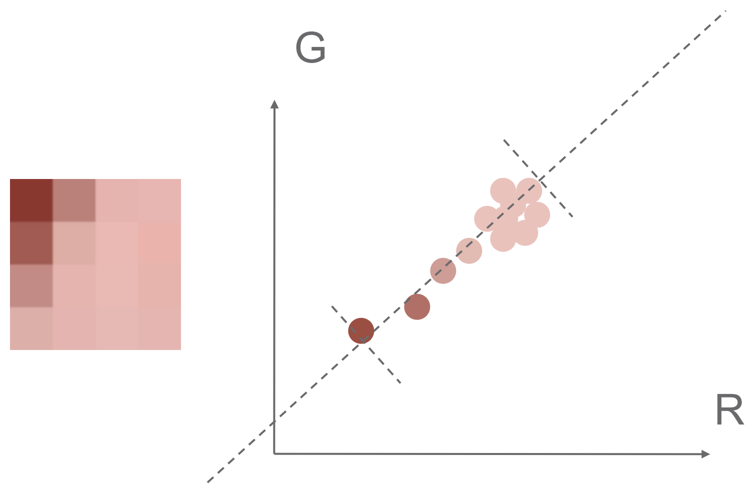
# S3TC – S3 Texture Compression

- Once we have the major axis, it is simply an issue of placing the two outer colors
- One way is to project the colors onto the line, and use the end points.



# S3TC – S3 Texture Compression

- Once we have the major axis, it is simply an issue of placing the two outer colors
- One way is to project the colors onto the line, and use the end points.
- The position can then be refined with linear search along the line.



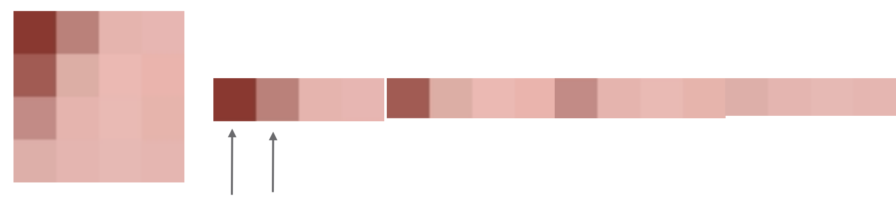
# S3TC – S3 Texture Compression

- Another approach, not dealing with PCA, is based on the assumption that the end colors should be close to some color in the block.

© Ericsson AB 2009 | Ericsson Internal | X (X) | Date

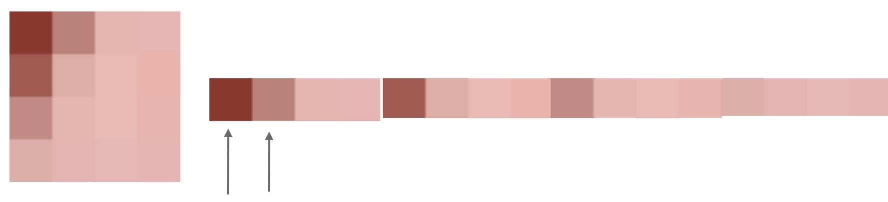
# S3TC – S3 Texture Compression

- Another approach, not dealing with PCA, is based on the assumption that the end colors should be close to some color in the block.
- The algorithm is then to try every pair of colors in the block as end points, and compress the block. The block with the smallest error wins.



# S3TC – S3 Texture Compression

- Another approach, not dealing with PCA, is based on the assumption that the end colors should be close to some color in the block.
- The algorithm is then to try every pair of colors in the block as end points, and compress the block. The block with the smallest error wins.
- At most 15*16/2 = 128 trials.



# S3TC – S3 Texture Compression

- Another approach, not dealing with PCA, is based on the assumption that the end colors should be close to some color in the block.
- The algorithm is then to try every pair of colors in the block as end points, and compress the block. The block with the smallest error wins.
- At most 15*16/2 = 128 trials.

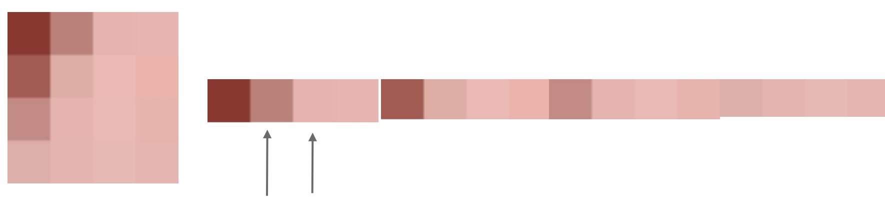
© Ericsson AB 2009 | Ericsson Internal | X (X) | Date

# S3TC – S3 Texture Compression

- Another approach, not dealing with PCA, is based on the assumption that the end colors should be close to some color in the block.
- The algorithm is then to try every pair of colors in the block as end points, and compress the block. The block with the smallest error wins.
- At most 15*16/2 = 128 trials.

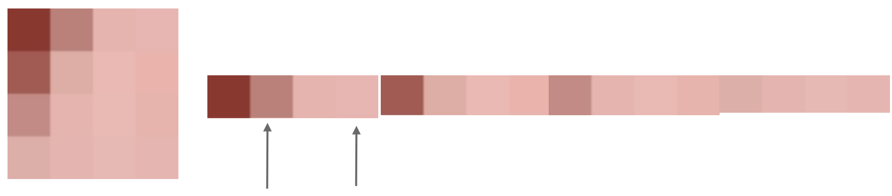
# S3TC – S3 Texture Compression

- Another approach, not dealing with PCA, is based on the assumption that the end colors should be close to some color in the block.
- The algorithm is then to try every pair of colors in the block as end points, and compress the block. The block with the smallest error wins.
- At most 15*16/2 = 128 trials.



# S3TC – S3 Texture Compression

- Another approach, not dealing with PCA, is based on the assumption that the end colors should be close to some color in the block.
- The algorithm is then to try every pair of colors in the block as end points, and compress the block. The block with the smallest error wins.
- At most 15*16/2 = 128 trials.



# S3TC – S3 Texture Compression

- Another approach, not dealing with PCA, is based on the assumption that the end colors should be close to some color in the block.
- The algorithm is then to try every pair of colors in the block as end points, and compress the block. The block with the smallest error wins.
- At most 15*16/2 = 128 trials.

# PVR-TC

# PVR-TC

- PVR-TC by Fenney builds on the fact that a down-sampled, up-scaled image is rather similar to itself.

# PVR-TC

- PVR-TC by Fenney builds on the fact that a down-sampled, up-scaled image is rather similar to itself.

# PVR-TC

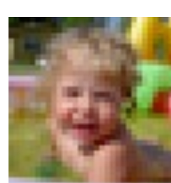
› PVR-TC by Fenney builds on the fact that a down-sampled, up-scaled image is rather similar to itself.

down-scale

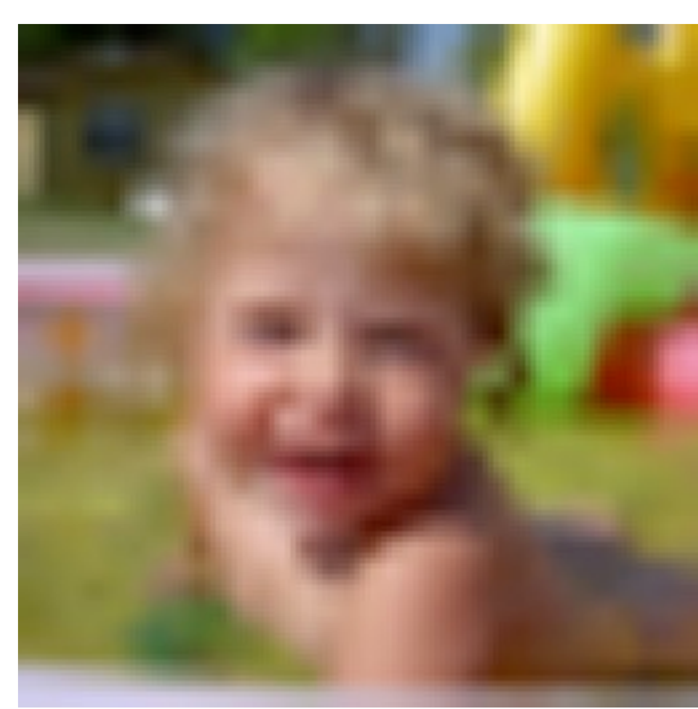
# PVR–TC

› PVR-TC by Fenney builds on the fact that a down-sampled, up-scaled image is rather similar to itself.

down-scale →



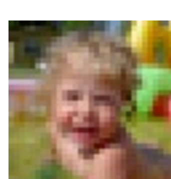
up-scale →



# PVR–TC

- PVR-TC by Fenney builds on the fact that a down-sampled, up-scaled image is rather similar to itself.
- The only thing that is missing is sharp edges.

down-
scale



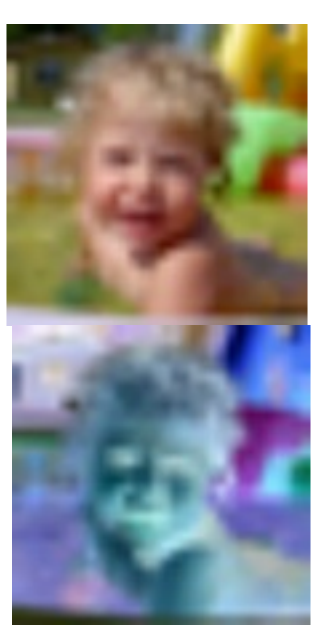
up-
scale

# PVR-TC

- Fenney solves this by having two low resolution images, and a bitmask.

# PVR-TC

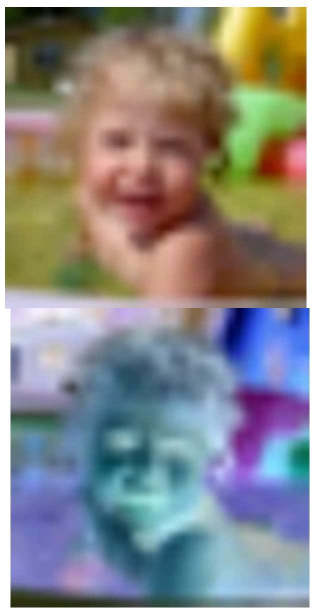
- Fenney solves this by having two low resolution images, and a bitmask.



| | | | | | |
|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 |

# PVR-TC

- Fenney solves this by having two low resolution images, and a bitmask.
- Each pixel can then choose which image it wants to take its color from.



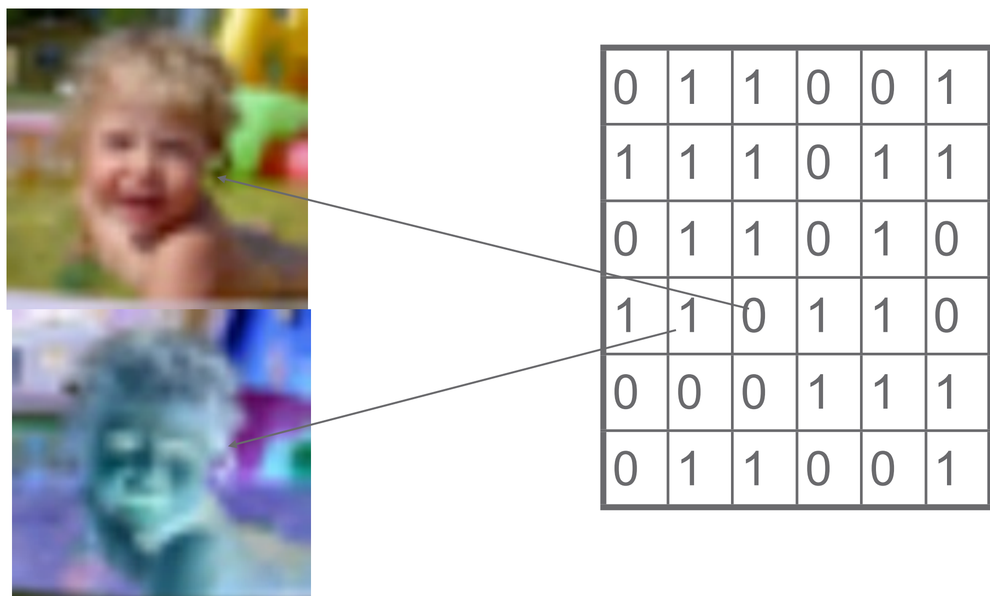
| 0 | 1 | 1 | 0 | 0 | 1 |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 |

# PVR-TC

- Fenney solves this by having two low resolution images, and a bitmask.
- Each pixel can then choose which image it wants to take its color from.

# PVR–TC

- Fenney solves this by having two low resolution images, and a bitmask.
- Each pixel can then choose which image it wants to take its color from.



| 0 | 1 | 1 | 0 | 0 | 1 |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 |

# PVR-TC

- Fenney solves this by having two low resolution images, and a bitmask.
- Each pixel can then choose which image it wants to take its color from.

| 0 | 1 | 1 | 0 | 0 | 1 |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 |

(fake)

# PVR-TC

› For instance, if one of the two images is completely white, then one can get perfectly sharp white text over the other image by arranging the bit mask.

| 0 | 0 | 0 | 0 | 0 | 1 |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 |

(fake)

Sharp text

# PVR-TC

- By making the bitmask contain four levels (2 bits per pixel), Fenney can blend between the first and the second image.

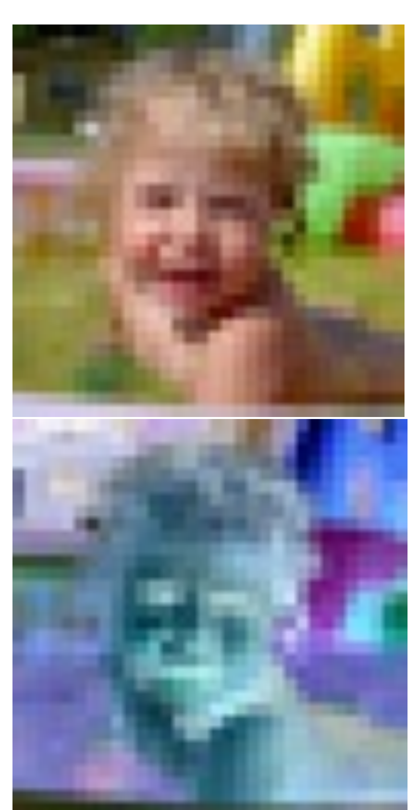
| 0 | 0 | 0 | 0 | 2 | 3 |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 3 | 3 | 3 |
| 0 | 3 | 0 | 0 | 2 | 3 |
| 3 | 1 | 0 | 0 | 2 | 2 |
| 3 | 1 | 0 | 0 | 3 | 3 |
| 1 | 3 | 3 | 3 | 2 | 3 |

(fake)

Sharp

# PVR-TC

- Each block in PVR-TC includes one color from each low-resolution image in RGB565



# PVR-TC

- Each block in PVR-TC includes one color from each low-resolution image in RGB565

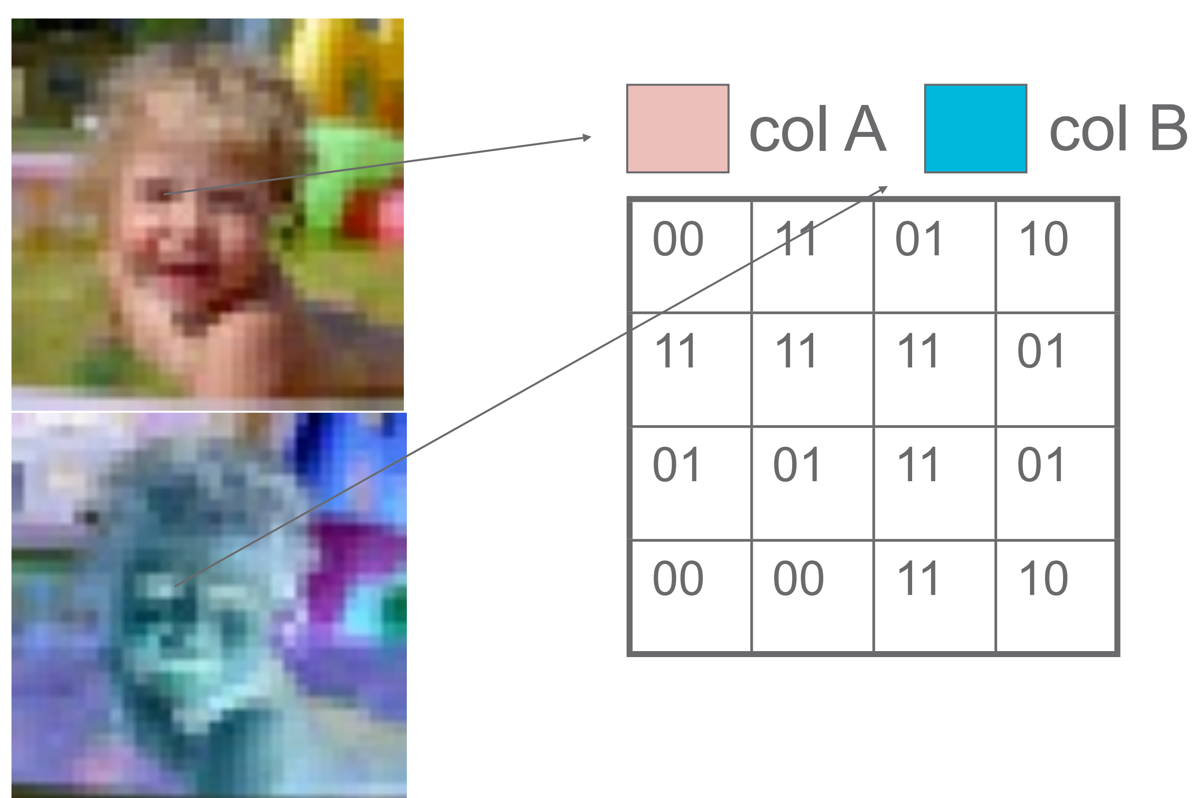
# PVR-TC

- Each block in PVR-TC includes one color from each low-resolution image in RGB565

col A col B

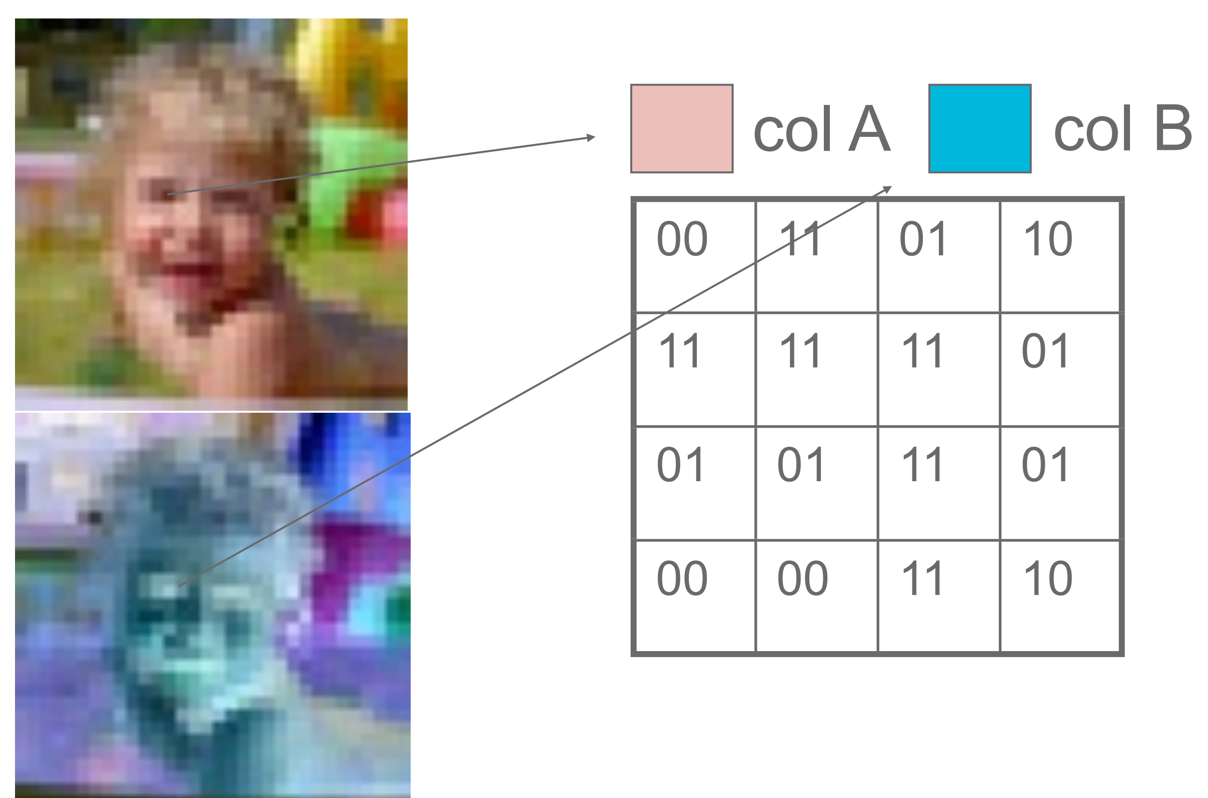
# PVR-TC

› Each block in PVR-TC includes one color from each low-resolution image in RGB565, plus the bitmask.



# PVR–TC

- Each block in PVR-TC includes one color from each low-resolution image in RGB565, plus the bitmask.




- This means 16+16+32=64 bits per block, or 4 bpp.

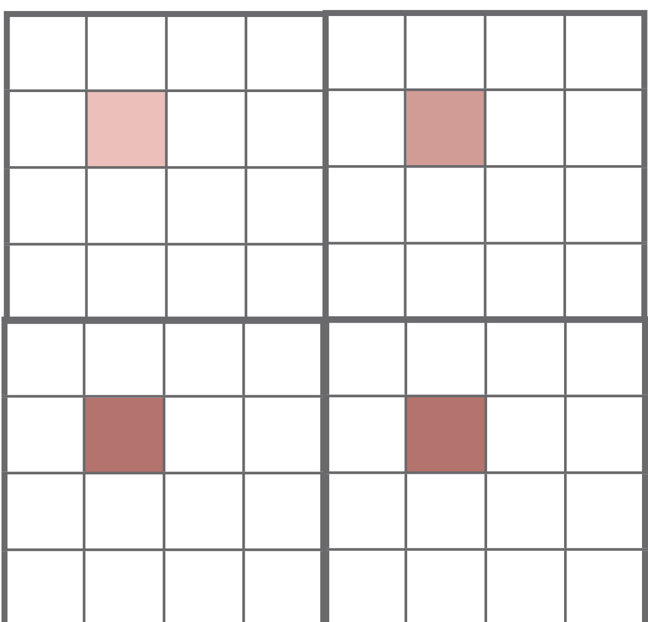
# PVR-TC

- The colors are situated in the top left middle pixel.

© Ericsson AB 2009 | Ericsson Internal | X (X) | Date

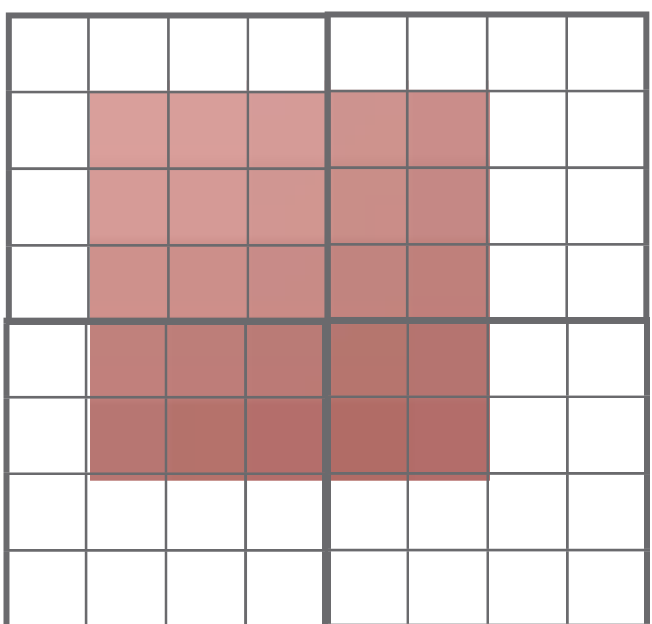
# PVR-TC

- The colors are situated in the top left middle pixel.
- To decode a block, the neighboring blocks are needed.



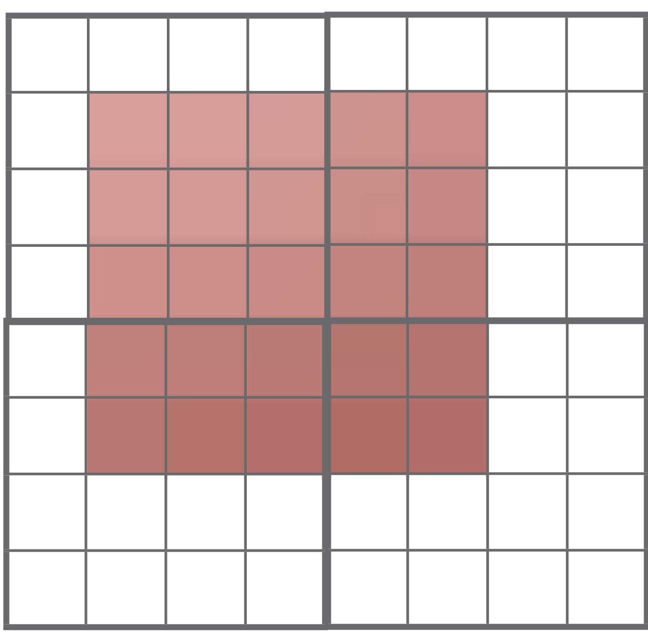
# PVR-TC

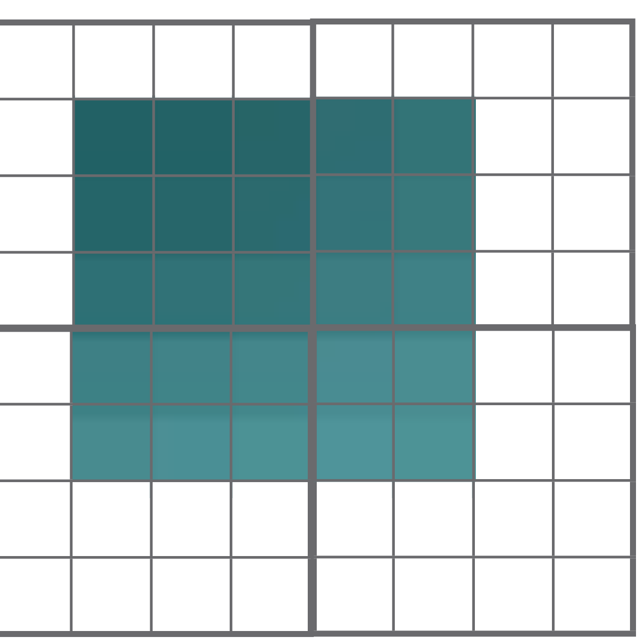
- The colors are situated in the top left middle pixel.
- To decode a block, the neighboring blocks are needed.
- Bilinear upscaling is used.



# PVR-TC

› The same thing is done for color B





# PVR-TC

- The bit mask is now used to choose between the two.

00

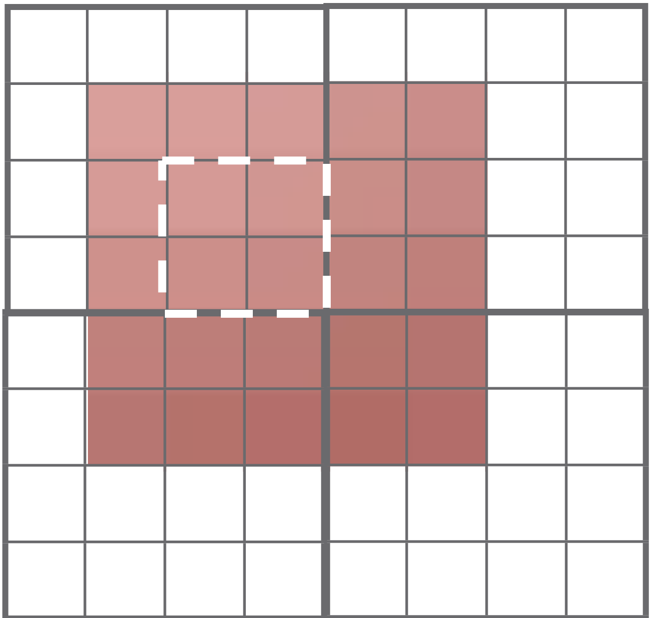
01

10

11

# PVR-TC

- Even though surrounding blocks must be read, it is never necessary to load more than four blocks to decode an entire 2x2 area.
- This is the same as the worst case for S3TC etc.



# PACKMAN

# PACKMAN

## texture compression for mobile phones

| Scheme | Complexity | Quality |
|---|---|---|
| CCC [Campbell et al. '86] | Low – but uses indirect addressing | Medium/Low |

# PACKMAN

## texture compression for mobile phones

| Scheme | Complexity | Quality |
|---|---|---|
| CCC [Campbell et al. ’86] | Low – but uses indirect addressing | Medium/Low |
| S3TC/DXTC [Iourcha et al. ‘99] | Medium/High – performs multiplication with 1/3 and 2/3 | High |

# PACKMAN

## texture compression for mobile phones

| Scheme | Complexity | Quality |
|---|---|---|
| CCC [Campbell et al. '86] | Low – but uses indirect addressing | Medium/Low |
| S3TC/DXTC [Iourcha et al. '99] | Medium/High – performs multiplication with 1/3 and 2/3 | High |
| PVR-TC [Fenney '03] | Medium/High – bilinear upscaling | High |

© Ericsson AB 2009 | Ericsson Internal |

texture compression for mobile phones

| Scheme | Complexity | Quality |
|---|---|---|
| CCC [Campbell et al. ’86] | Low – but uses indirect addressing | Medium/Low |
| S3TC/DXTC [Iourcha et al. ‘99] | Medium/High – performs multiplication with 1/3 and 2/3 | High |
| PVR-TC [Fenney ’03] | Medium/High – bilinear upscaling | High |
| ??? | Low | High |

# Design Goals

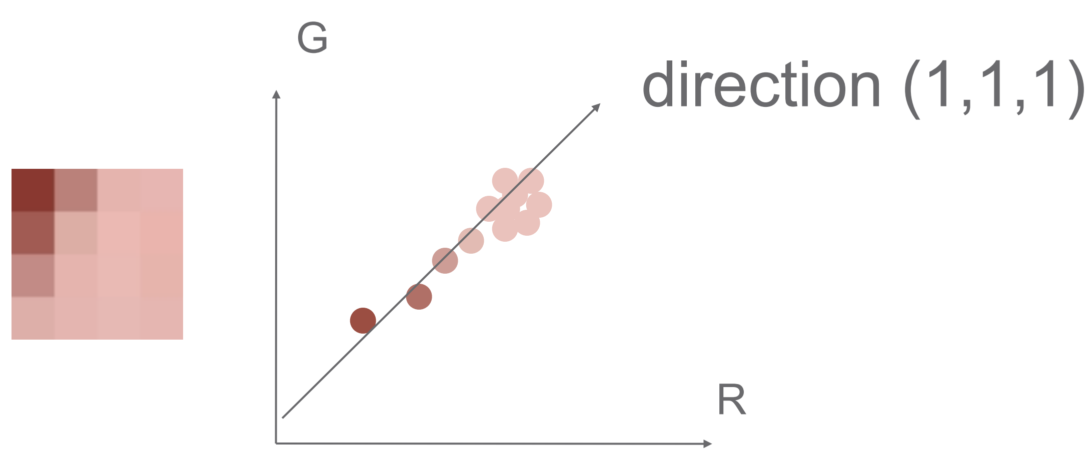
- Low Decompression Complexity
  - 8 parallel units needed for one trilinear operation per clock
  - mobile devices have very little surface area to spare
- High Image Quality
  - Should be on par with, or better than, industry standard DXTC at the same bit rate
- Should be “system friendly”
  - You want to be able to store compressed data in the cache, and that means that the decompression needs to be simple and fast.
  - No indirect data such as a color palette that increases latency
  - No data from adjacent blocks should be needed
  - For systems without a texture cache, a block size of 32 bits would be preferable, matching the size of the bus.

# Basic Idea PACKMAN

- The model that colors are along a line in RGB space has worked well for S3TC.
- Maybe we can pre-specify a specific direction in RGB space and thus save one color?
- The most common direction should be (1,1,1), that is, going from dark to bright.

# Basic Idea PACKMAN

- The model that colors are along a line in RGB space has worked well for S3TC.
- Maybe we can pre-specify a specific direction in RGB space and thus save one color?
- The most common direction should be (1,1,1), that is, going from dark to bright.

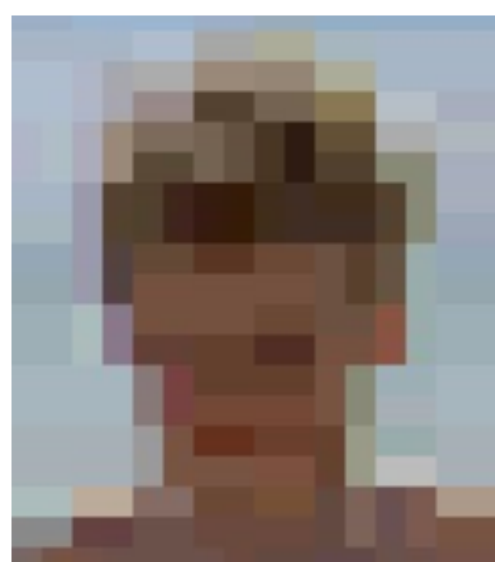


# Basic Idea PACKMAN

- In addition, the Human Visual System is more sensitive to luminance than to chrominance
- In video and still image coding, chrominance information is most often subsampled in the x- and y- direction (MPEG, JPEG, H263, H264 etc). Loosely speaking, chrominance is defined per 2x2 block.
- PACKMAN has basically only one color per 2x4 block. The rest is luminance information
- Code each 2x4 block using 32 bits

# Basic Idea PACKMAN

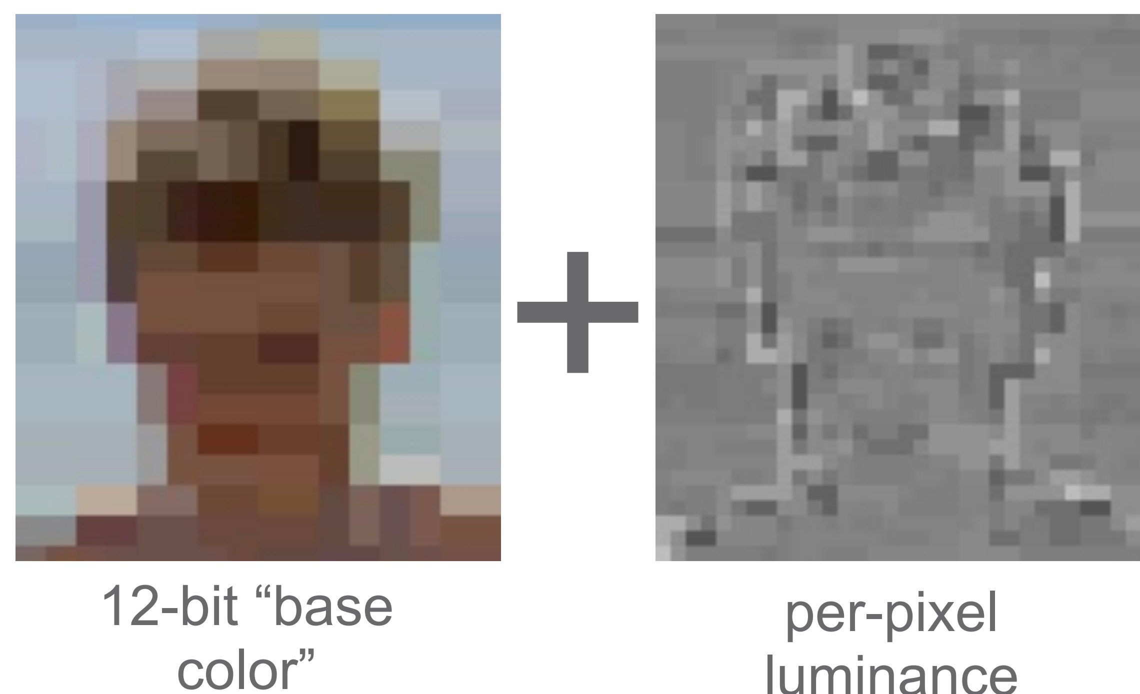
› Use only 12 bits to specify a “base color” for a 2x4 block



12-bit “base color”

# Basic Idea PACKMAN

- Use only 12 bits to specify a “base color” for a 2x4 block
- Modify the luminance for each pixel in the block



12-bit “base color”

per-pixel luminance

# Basic Idea PACKMAN

- Use only 12 bits to specify a “base color” for a 2x4 block
- Modify the luminance for each pixel in the block

12-bit “base color” + per-pixel luminance = resulting image

# Luminance modification

- Only one value per pixel needed to specify luminance

# Luminance modification

- Only one value per pixel needed to specify luminance

R = 17
G = 34
B = 204

Base Color

# Luminance modification

- Only one value per pixel needed to specify luminance

R = 17 +110
G = 34 +110
B = 204 +110

Base Color

Add same value

# Luminance modification

- Only one value per pixel needed to specify luminance

| | | |
|---|---|---|
| R = 17 | +110 | = 127 |
| G = 34 | +110 | = 144 |
| B = 204 | +110 | = 255 (after clamping) |

Add same value

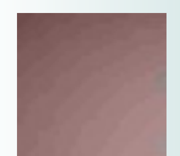
Resulting Color

# How to Specify Luminance

- Two bits per pixel are used to specify the luminance. Modifier is one out of four values.
- Problem: Small values [-8, -2, 2, 8]

# How to Specify Luminance

- Two bits per pixel are used to specify the luminance. Modifier is one out of four values.
- Problem: Small values [-8, -2, 2, 8]
  - smooth transitions OK



# How to Specify Luminance

- Two bits per pixel are used to specify the luminance. Modifier is one out of four values.
- Problem: Small values [-8, -2, 2, 8]
  - smooth transitions OK
  - sharp edges bad

# How to Specify Luminance

- Two bits per pixel are used to specify the luminance. Modifier is one out of four values.
- Problem: Small values [-8, -2, 2, 8]
  - smooth transitions OK
  - sharp edges bad
- Big values [-255, -127, 127, 255]
  - sharp edges OK

# How to Specify Luminance

- Two bits per pixel are used to specify the luminance. Modifier is one out of four values.
- Problem: Small values [-8, -2, 2, 8]
  - smooth transitions OK
  - sharp edges bad
- Big values [-255, -127, 127, 255]
  - sharp edges OK
  - smooth transitions bad

# How to Specify Luminance

- Two bits per pixel are used to specify the luminance. Modifier is one out of four values.
- Problem: Small values [-8, -2, 2, 8]
  - smooth transitions OK
  - sharp edges bad
- Big values [-255, -127, 127, 255]
  - sharp edges OK
  - smooth transitions bad
- Solution: Codebook of tables, one/block.

# Modifier Codebook

- We created the codebook from random numbers by minimizing the error for a set of images.

| ?? | ?? | ?? | ?? |
|---|---|---|---|
| ?? | ?? | ?? | ?? |
| ?? | ?? | ?? | ?? |
| ?? | ?? | ?? | ?? |
| ?? | ?? | ?? | ?? |
| ?? | ?? | ?? | ?? |
| ?? | ?? | ?? | ?? |
| ?? | ?? | ?? | ?? |
| ?? | ?? | ?? | ?? |
| ?? | ?? | ?? | ?? |
| ?? | ?? | ?? | ?? |
| ?? | ?? | ?? | ?? |
| ?? | ?? | ?? | ?? |
| ?? | ?? | ?? | ?? |
| ?? | ?? | ?? | ?? |
| ?? | ?? | ?? | ?? |

# Modifier Codebook

- We created the codebook from random numbers by minimizing the error for a set of images.

| | | | |
|---|---|---|---|
| -8 | -2 | 2 | 8 |
| -12 | -4 | 4 | 12 |
| -16 | -4 | 4 | 16 |
| -24 | -8 | 8 | 24 |
| -31 | -6 | 6 | 31 |
| -34 | -12 | 12 | 34 |
| -47 | -19 | 19 | 47 |
| -50 | -8 | 8 | 50 |
| -62 | -12 | 12 | 62 |
| -68 | -24 | 24 | 68 |
| -80 | -28 | 28 | 80 |
| -94 | -38 | 38 | 94 |
| -100 | -16 | 16 | 100 |
| -127 | -42 | 42 | 127 |
| -160 | -56 | 56 | 160 |
| -254 | -84 | 84 | 254 |

# Modifier Codebook

- We created the codebook from random numbers by minimizing the error for a set of images.
  - Simulated Annealing
  - Modified version of LBG-algorithm
- Symmetry was enforced to reduce on-chip memory

| | | | |
|---|---|---|---|
| -8 | -2 | 2 | 8 |
| -12 | -4 | 4 | 12 |
| -16 | -4 | 4 | 16 |
| -24 | -8 | 8 | 24 |
| -31 | -6 | 6 | 31 |
| -34 | -12 | 12 | 34 |
| -47 | -19 | 19 | 47 |
| -50 | -8 | 8 | 50 |
| -62 | -12 | 12 | 62 |
| -68 | -24 | 24 | 68 |
| -80 | -28 | 28 | 80 |
| -94 | -38 | 38 | 94 |
| -100 | -16 | 16 | 100 |
| -127 | -42 | 42 | 127 |
| -160 | -56 | 56 | 160 |
| -254 | -84 | 84 | 254 |

# Modifier Codebook

- We created the codebook from random numbers by minimizing the error for a set of images.
  - Simulated Annealing
  - Modified version of LBG-algorithm
- Symmetry was enforced to reduce on-chip memory

| | | | |
|---|---|---|---|
| -8 | -2 | 2 | 8 |
| -12 | -4 | 4 | 12 |
| -16 | -4 | 4 | 16 |
| -24 | -8 | 8 | 24 |
| -31 | -6 | 6 | 31 |
| -34 | -12 | 12 | 34 |
| -47 | -19 | 19 | 47 |
| -50 | -8 | 8 | 50 |
| -62 | -12 | 12 | 62 |
| -68 | -24 | 24 | 68 |
| -80 | -28 | 28 | 80 |
| -94 | -38 | 38 | 94 |
| -100 | -16 | 16 | 100 |
| -127 | -42 | 42 | 127 |
| -160 | -56 | 56 | 160 |
| -254 | -84 | 84 | 254 |

# Modifier Codebook

- We created the codebook from random numbers by minimizing the error for a set of images.
  - Simulated Annealing
  - Modified version of LBG-algorithm
- Symmetry was enforced to reduce on-chip memory
- This way only half the table needed to be stored on chip.

| | | | |
|---|---|---|---|
| -8 | -2 | 2 | 8 |
| -12 | -4 | 4 | 12 |
| -16 | -4 | 4 | 16 |
| -24 | -8 | 8 | 24 |
| -31 | -6 | 6 | 31 |
| -34 | -12 | 12 | 34 |
| -47 | -19 | 19 | 47 |
| -50 | -8 | 8 | 50 |
| -62 | -12 | 12 | 62 |
| -68 | -24 | 24 | 68 |
| -80 | -28 | 28 | 80 |
| -94 | -38 | 38 | 94 |
| -100 | -16 | 16 | 100 |
| -127 | -42 | 42 | 127 |
| -160 | -56 | 56 | 160 |
| -254 | -84 | 84 | 254 |

# Modifier Codebook

- We created the codebook from random numbers by minimizing the error for a set of images.
  - Simulated Annealing
  - Modified version of LBG-algorithm
- Symmetry was enforced to reduce on-chip memory
- This way only half the table needed to be stored on chip.

| | |
|---|---|
| -8 | -2 |
| -12 | -4 |
| -16 | -4 |
| -24 | -8 |
| -31 | -6 |
| -34 | -12 |
| -47 | -19 |
| -50 | -8 |
| -62 | -12 |
| -68 | -24 |
| -80 | -28 |
| -94 | -38 |
| -100 | -16 |
| -127 | -42 |
| -160 | -56 |
| -254 | -84 |

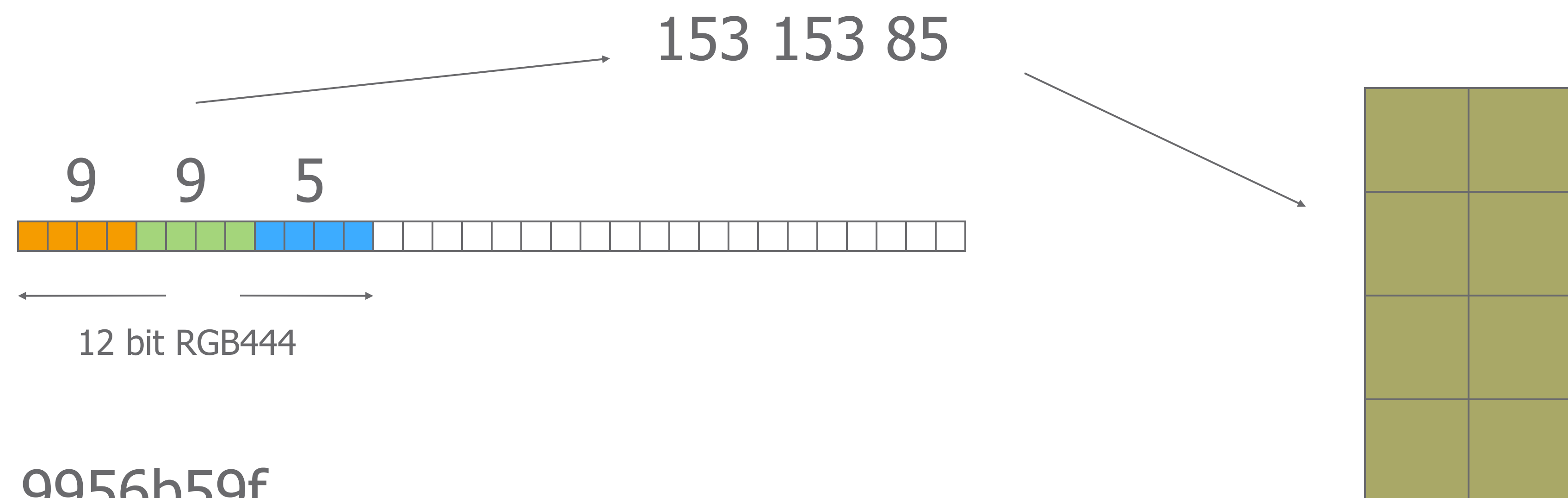
# Modifier Codebook

- We created the codebook from random numbers by minimizing the error for a set of images.
  - Simulated Annealing
  - Modified version of LBG-algorithm
- Symmetry was enforced to reduce on-chip memory
- This way only half the table needed to be stored on chip.
- The same table is used for all textures – can be hardwired into the logic.

| | |
|---|---|
| -8 | -2 |
| -12 | -4 |
| -16 | -4 |
| -24 | -8 |
| -31 | -6 |
| -34 | -12 |
| -47 | -19 |
| -50 | -8 |
| -62 | -12 |
| -68 | -24 |
| -80 | -28 |
| -94 | -38 |
| -100 | -16 |
| -127 | -42 |
| -160 | -56 |
| -254 | -84 |

# Bit outline

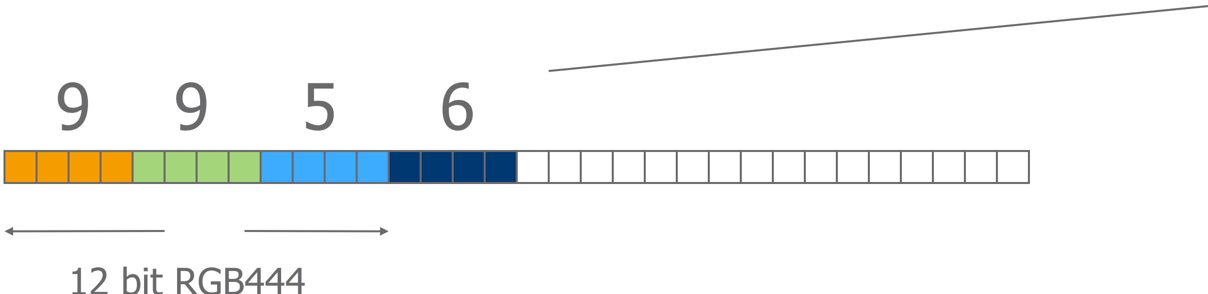
- First 12 bits is RGB444 which gives the base color for the entire block.



# Bit outline

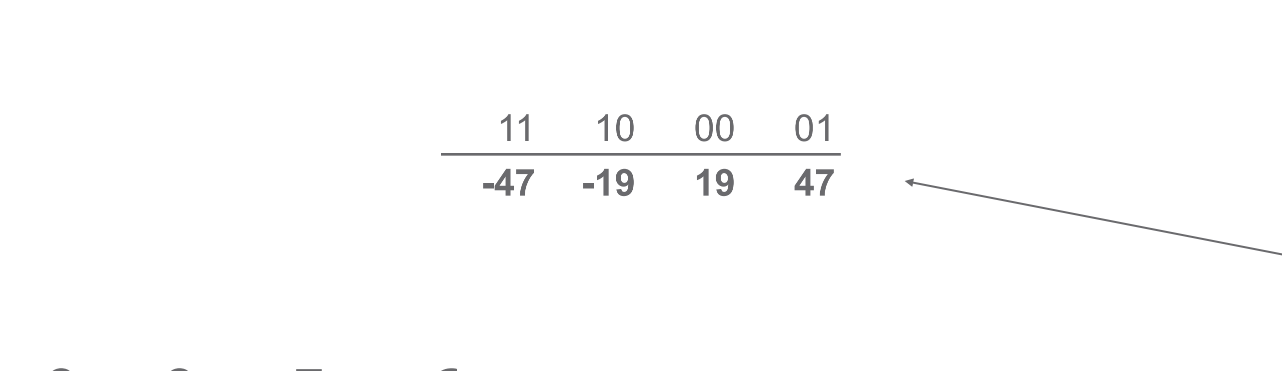
- Next 4 bits selects a table from a set of 16 tables

| 11 | 10 | 00 | 01 |
|---|---|---|---|
| -8 | -2 | 2 | 8 |
| -12 | -4 | 4 | 12 |
| -16 | -4 | 4 | 16 |
| -24 | -8 | 8 | 24 |
| -31 | -6 | 6 | 31 |
| -34 | -12 | 12 | 34 |
| -47 | -19 | 19 | 47 |
| -50 | -8 | 8 | 50 |
| -62 | -12 | 12 | 62 |
| -68 | -24 | 24 | 68 |
| -80 | -28 | 28 | 80 |
| -94 | -38 | 38 | 94 |
| -100 | -16 | 16 | 100 |
| -127 | -42 | 42 | 127 |
| -160 | -56 | 56 | 160 |
| -254 | -84 | 84 | 254 |



# Bit outline

› Next 4 bits selects a table from a set of 16 tables.



| 11 | 10 | 00 | 01 |
|---|---|---|---|
| **-47** | **-19** | **19** | **47** |

| 11 | 10 | 00 | 01 |
|---|---|---|---|
| -8 | -2 | 2 | 8 |
| -12 | -4 | 4 | 12 |
| -16 | -4 | 4 | 16 |
| -24 | -8 | 8 | 24 |
| -31 | -6 | 6 | 31 |
| -34 | -12 | 12 | 34 |
| -47 | -19 | 19 | 47 |
| -50 | -8 | 8 | 50 |
| -62 | -12 | 12 | 62 |
| -68 | -24 | 24 | 68 |
| -80 | -28 | 28 | 80 |
| -94 | -38 | 38 | 94 |
| -100 | -16 | 16 | 100 |
| -127 | -42 | 42 | 127 |
| -160 | -56 | 56 | 160 |
| -254 | -84 | 84 | 254 |

9 9 5 6

12 bit RGB444

# Bit outline

- The next 2 bits modifies the first pixel according to the table…

| 11 | 10 | 00 | 01 |
|---|---|---|---|
| **-47** | **-19** | **19** | **47** |

| | 153 | 153 | 85 |
|---|---|---|---|
| + | -19 | -19 | -19 |
| = | 134 | 134 | 66 |

9 9 5 6 10

12 bit RGB444

# Bit outline

- The next 2 bits modifies the first pixel according to the table… and so on.

| 11 | 10 | 00 | 01 |
|---|---|---|---|
| -47 | -19 | 19 | 47 |

| | 153 | 153 | 85 |
|---|---|---|---|
| + | -47 | -47 | -47 |
| = | 106 | 106 | 38 |

9 9 5 6 10 11

12 bit RGB444

# Bit outline

› The next 2 bits modifies the first pixel according to the table… and so on.

# Bit outline

- The next 2 bits modifies the first pixel according to the table… and so on.

| 11 | 10 | 00 | 01 |
|---|---|---|---|
| **-47** | **-19** | **19** | **47** |

| | | | |
|---|---|---|---|
| | 153 | 153 | 85 |
| + | 47 | 47 | 47 |
| = | 200 | 200 | 132 |

9 9 5 6 10 11 01 01

12 bit RGB444

# Bit outline

- The next 2 bits modifies the first pixel according to the table… and so on.

| 11 | 10 | 00 | 01 |
|---|---|---|---|
| **-47** | **-19** | **19** | **47** |

| | 153 | 153 | 85 |
|---|---|---|---|
| + | -19 | -19 | -19 |
| = | 134 | 134 | 66 |

9 9 5 6 10 11 01 01 10

12 bit RGB444

# Bit outline

- The next 2 bits modifies the first pixel according to the table… and so on.

| 11 | 10 | 00 | 01 |
|---|---|---|---|
| **-47** | **-19** | **19** | **47** |

| | 153 | 153 | 85 |
|---|---|---|---|
| + | 47 | 47 | 47 |
| = | 200 | 200 | 132 |

9 9 5 6 10 11 01 01 10 01

12 bit RGB444

# Bit outline

- The next 2 bits modifies the first pixel according to the table… and so on.

| 11 | 10 | 00 | 01 |
| --- | --- | --- | --- |
| **-47** | **-19** | **19** | **47** |

| | | | |
| --- | --- | --- | --- |
| | 153 | 153 | 85 |
| + | -47 | -47 | -47 |
| = | 106 | 106 | 38 |

9 9 5 6 10 11 01 01 10 01 11

12 bit RGB444

# Bit outline

- The next 2 bits modifies the first pixel according to the table… and so on.

| 11 | 10 | 00 | 01 |
|---|---|---|---|
| **-47** | **-19** | **19** | **47** |

| | | | |
|---|---|---|---|
| | 153 | 153 | 85 |
| + | -47 | -47 | -47 |
| = | 106 | 106 | 38 |

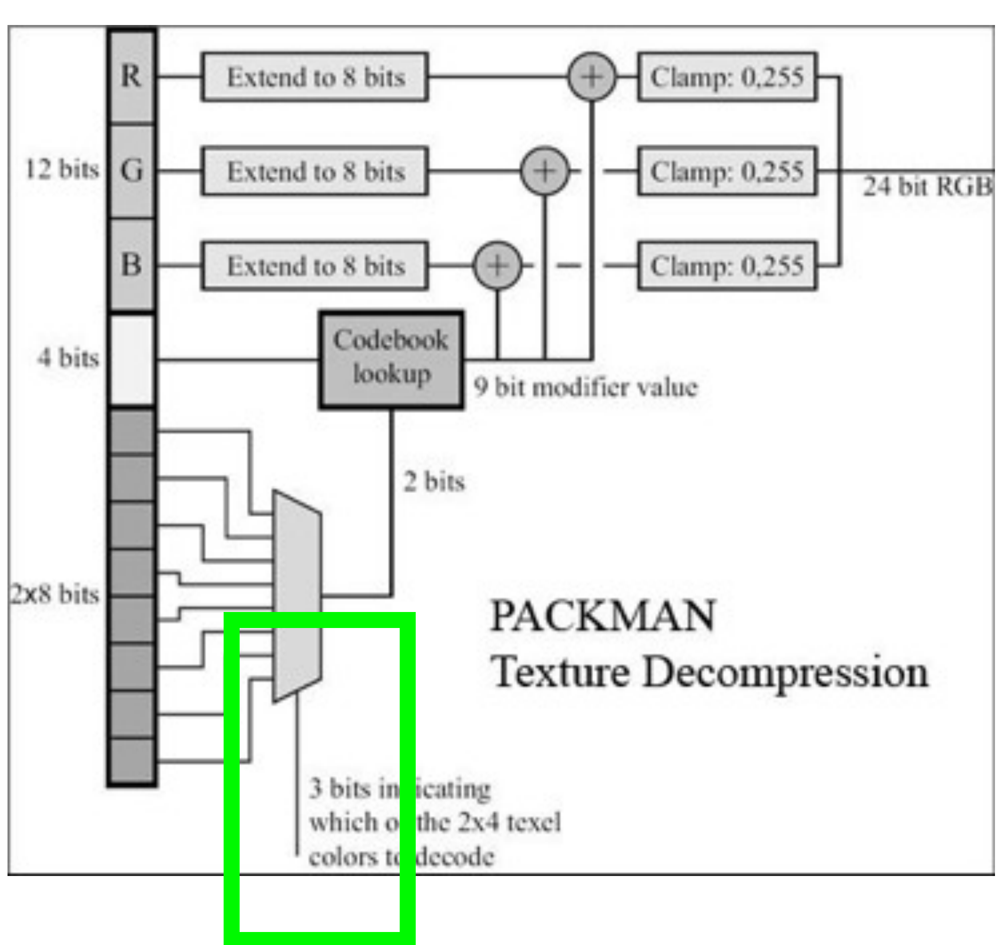
9 9 5 6 10 11 01 01 10 01 11 11

12 bit RGB444

© Ericsson AB 2009 | Ericsson Internal | X (X) | Date

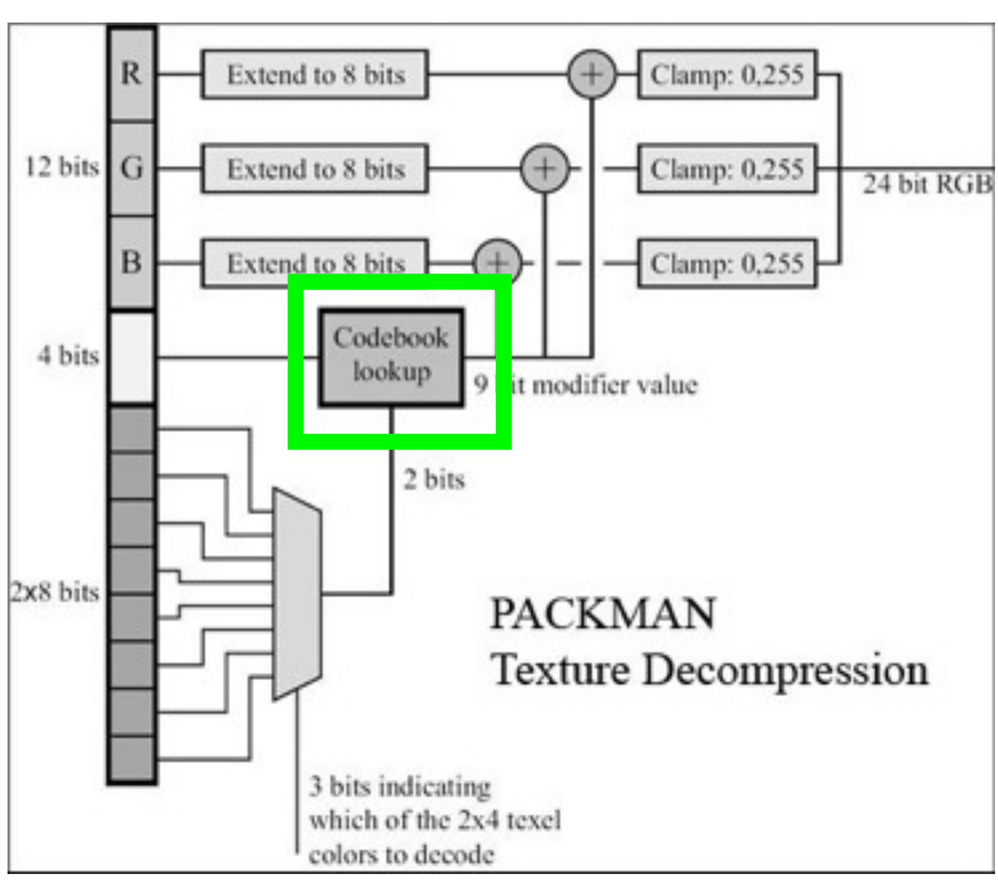
# Simple Decompression

- The correct texel is selected



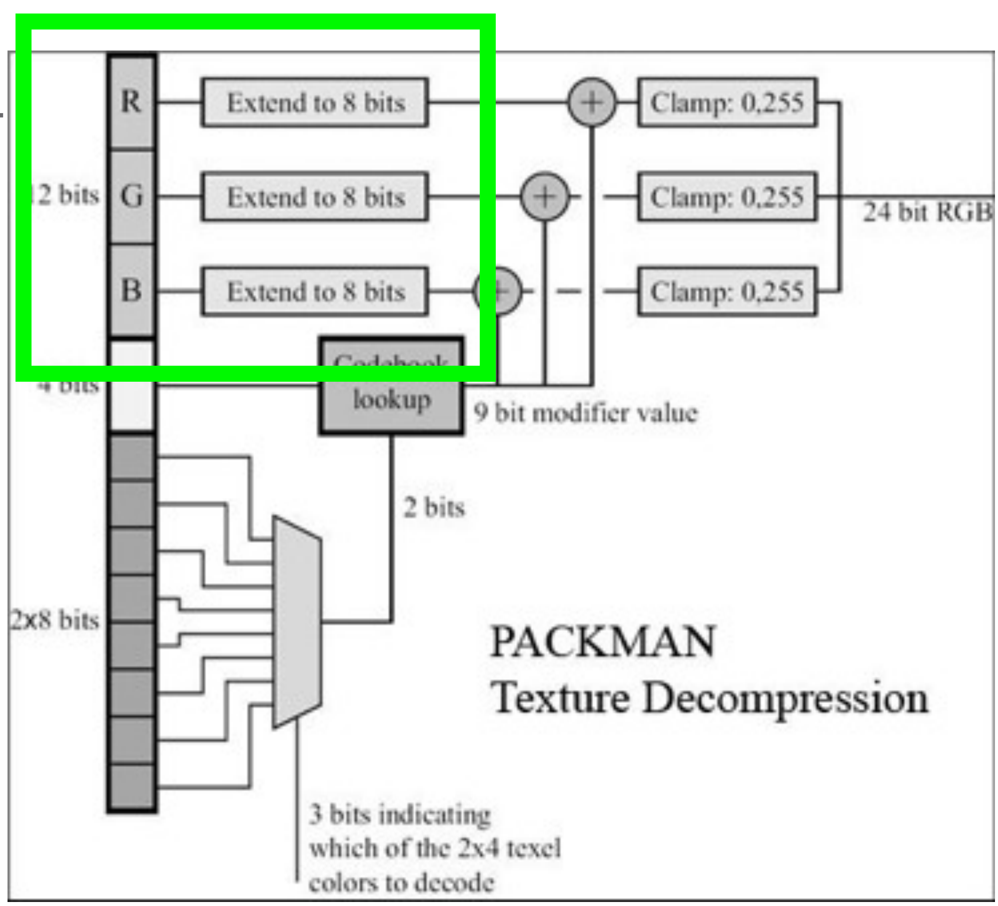
# Simple Decompression

- The correct texel is selected
- The modifier value is looked up



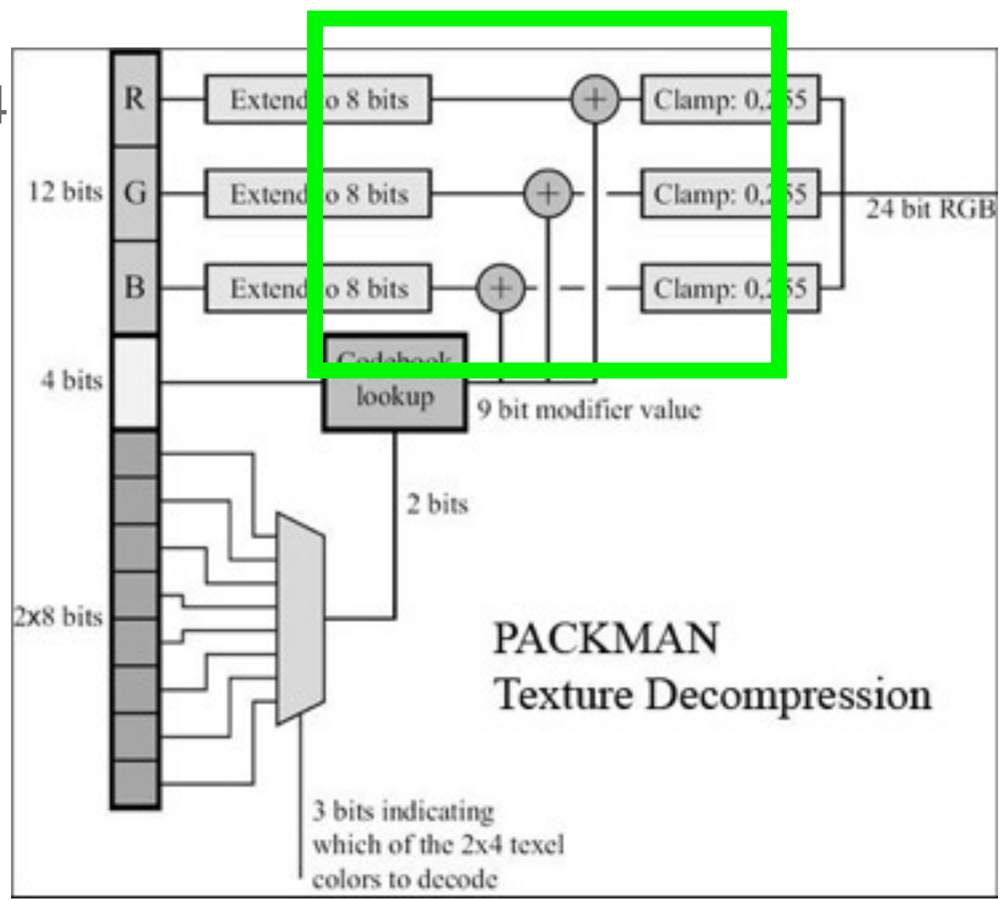
# Simple Decompression

- The correct texel is selected
- The modifier value is looked up
- The base color is extended to 24 bits



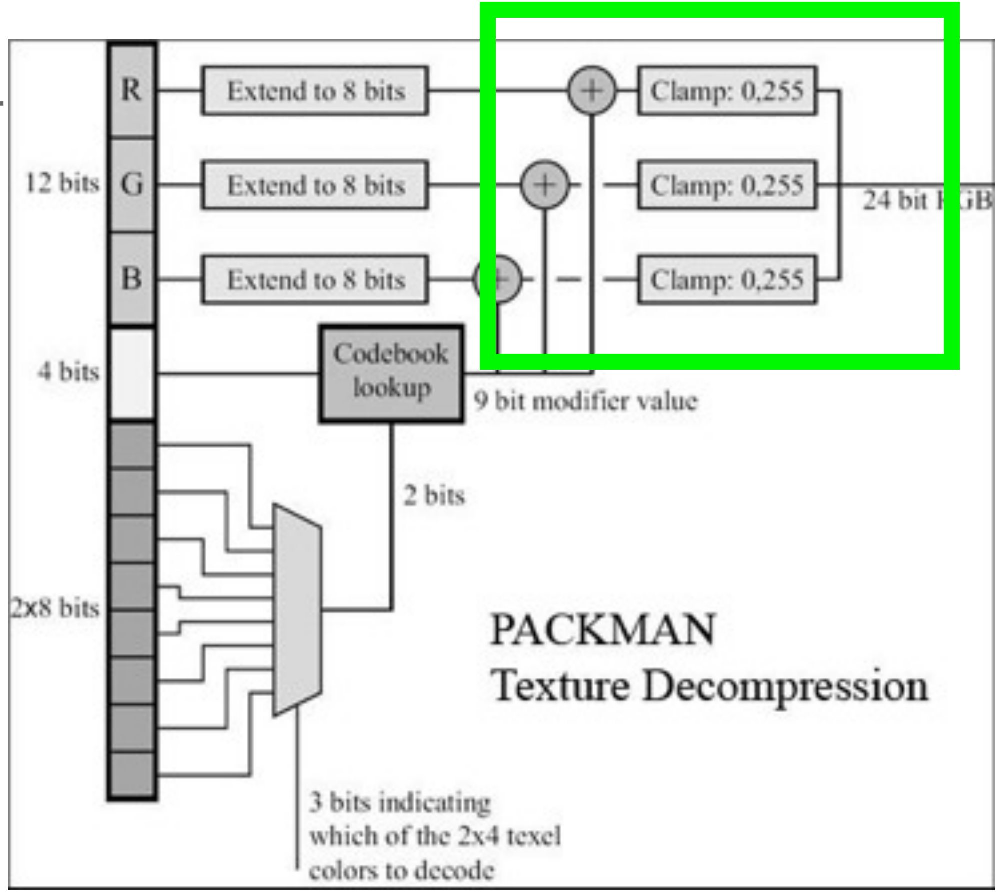
# Simple Decompression

› The correct texel is selected
› The modifier value is looked up
› The base color is extended to 24 bits
› The modifier value is added



# Simple Decompression

- The correct texel is selected
- The modifier value is looked up
- The base color is extended to 24 bits
- The modifier value is added
- The result is clamped



# ETC

Ericsson Texture Compression
(previously called iPACKMAN)

# PACKMAN Flaws

- PACKMAN was of very low complexity, but
  - 2 dB worse than DXTC in terms of Peak Signal to Noise Ratio (PSNR)
  - Suffered from chrominance banding / block artifacts due to low color resolution (RGB444)

original | PACKMAN

# PACKMAN Flaws

- PACKMAN was of very low complexity, but
  - 2 dB worse than DXTC in terms of Peak Signal to Noise Ratio (PSNR)
  - Suffered from chrominance banding / block artifacts due to low color resolution (RGB444)

original

PACKMAN

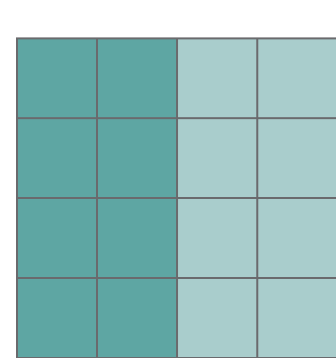
# Enhancing the Chrominance

...would increasing the block size help?

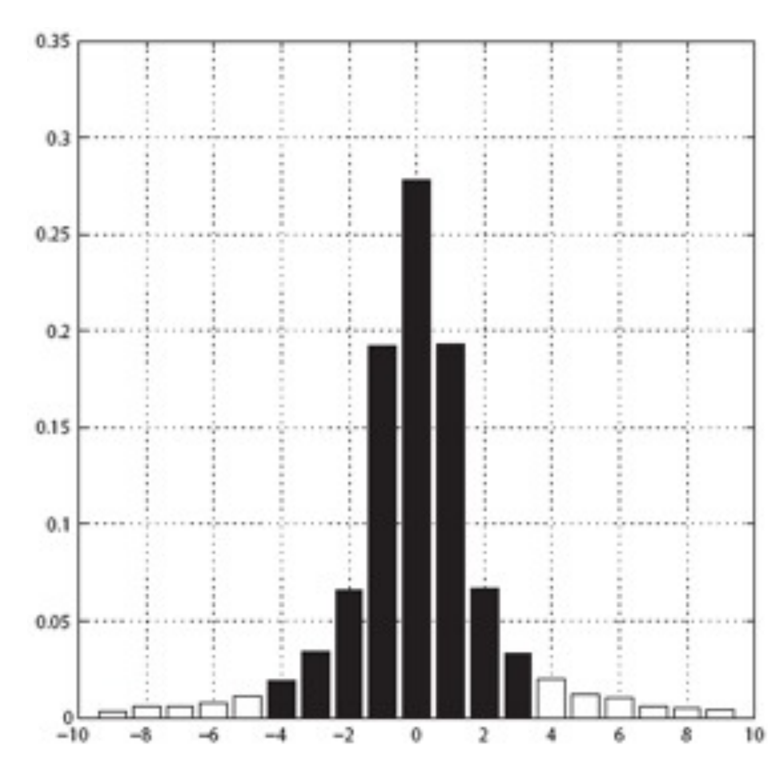
- By coding 4x4 blocks instead of 2x4 blocks, spatial redundancy could be better exploited.
- The small block size of 32 bits would be lost, but that was only beneficial in systems without a texture cache, so it was not a big loss.

# Neighboring Base Colors

## Quite Similar

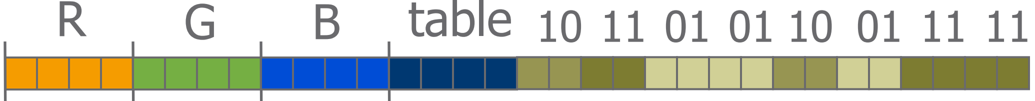


max(|R1–R2|, |G1–G2|, |B1–B2|)
(in RGB555)




88% within interval [–4, 3]

# Differential Encoding

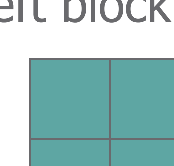


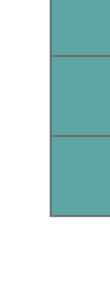
# Differential Encoding

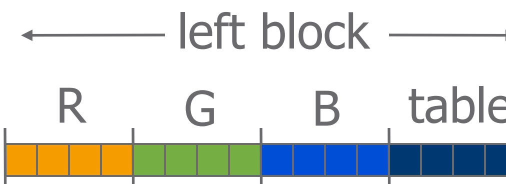
© Ericsson AB 2009 | Ericsson Internal | X (X) | Date

# Differential Encoding

- Instead of coding the left block with RGB444…

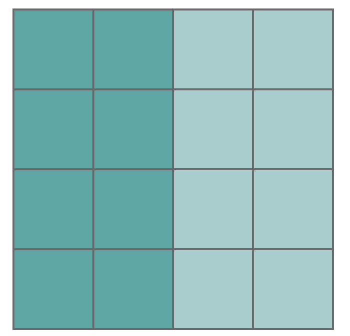


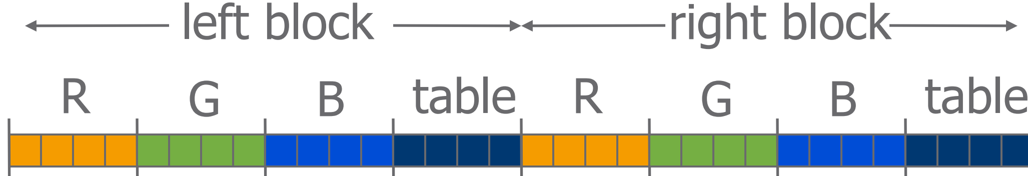





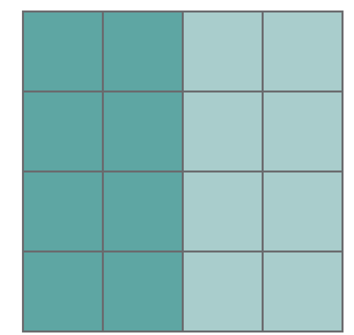
# Differential Encoding

- Instead of coding the left block with RGB444… and the right with RGB444…

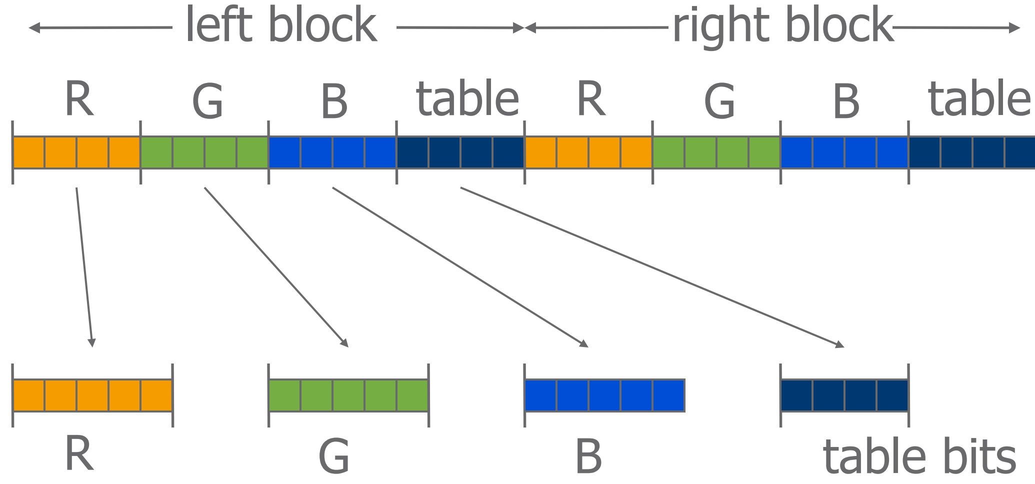


# Differential Encoding

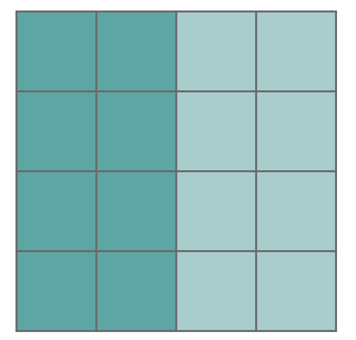


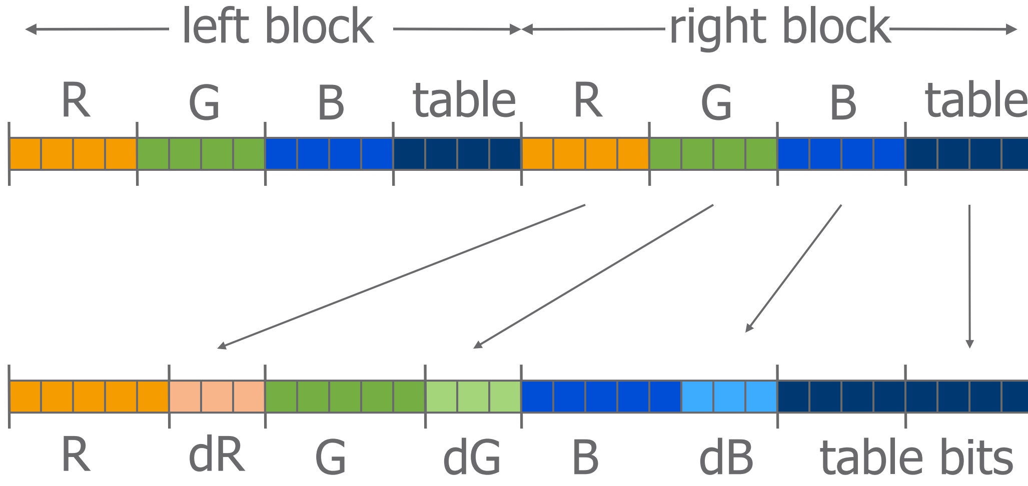
- Instead of coding the left block with RGB444… and the right with RGB444…
- We code the left with RGB555…



# Differential Encoding

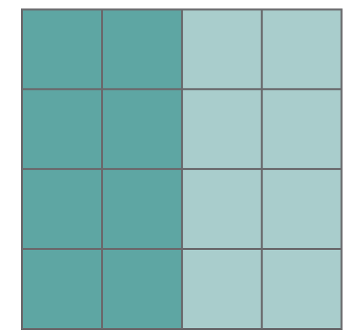
- Instead of coding the left block with RGB444… and the right with RGB444…
- We code the left with RGB555… and the right with dR dG dB 333.

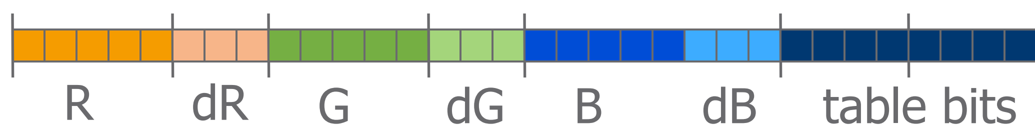


# Differential Encoding

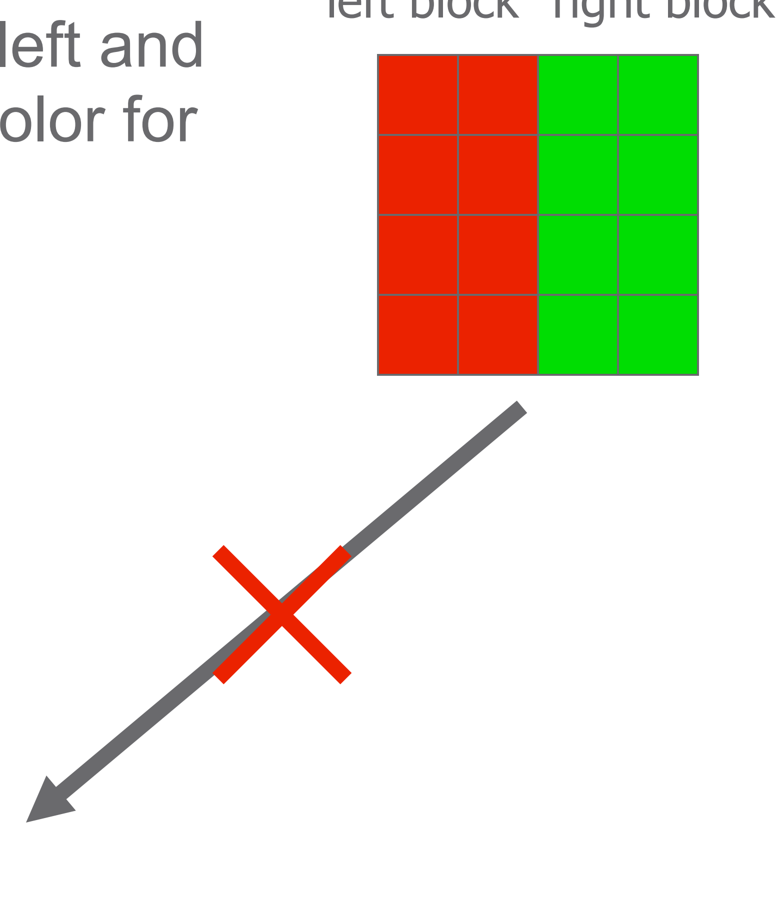
› However, in 10% of the cases, the left and right blocks will differ too much in color for differential coding.



# Differential Encoding

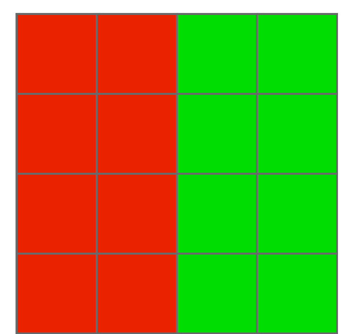
- However, in 10% of the cases, the left and right blocks will differ too much in color for differential coding.

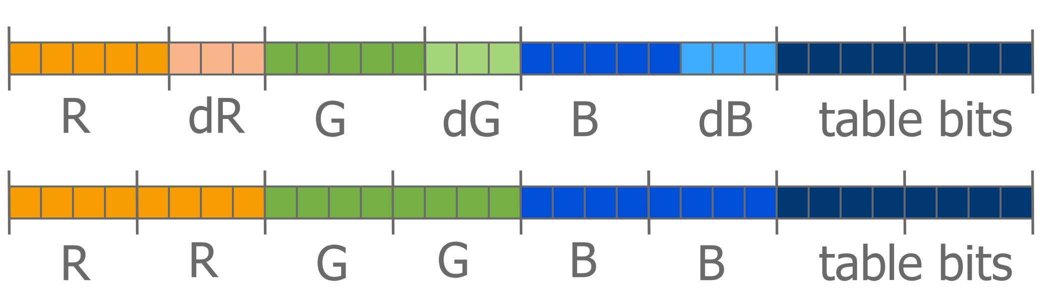



R dR G dG B dB table bits

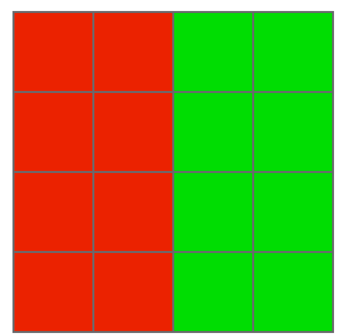
# Differential Encoding

- However, in 10% of the cases, the left and right blocks will differ too much in color for differential coding.
- For these blocks, we fall black to individual coding.

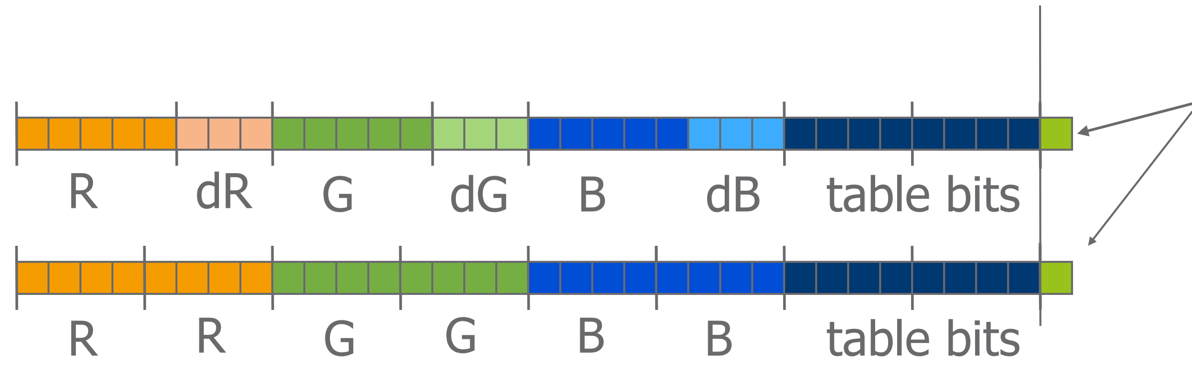


# Differential Encoding

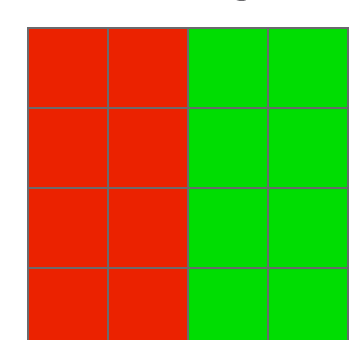



- However, in 10% of the cases, the left and right blocks will differ too much in color for differential coding.
- For these blocks, we fall black to individual coding.
- We thus need an extra bit to signal if we are in differential mode or not.

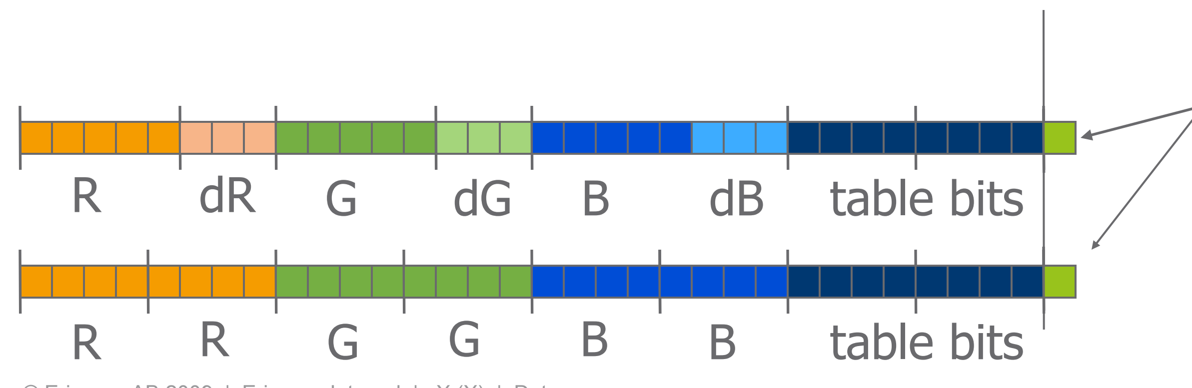



555differential or 444

# Differential Encoding




- However, in 10% of the cases, the left and right blocks will differ too much in color for differential coding.
- For these blocks, we fall black to individual coding.
- We thus need an extra bit to signal if we are in differential mode or not.
- We must take that bit from somewhere.

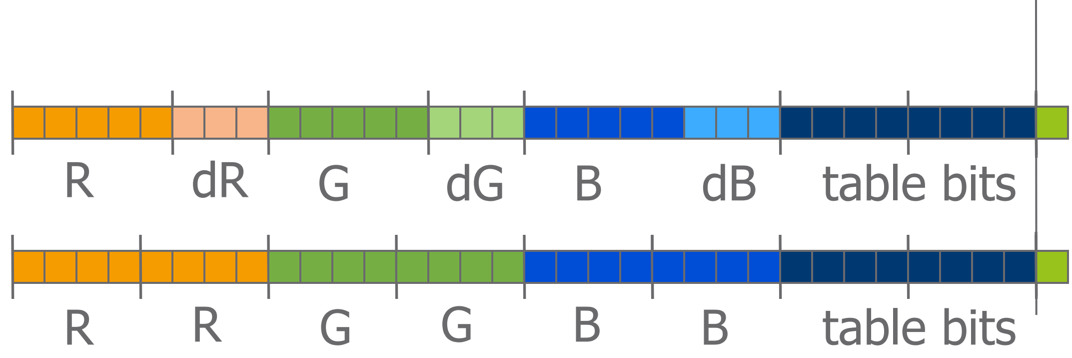



555differential or 444

# Differential Encoding

| -8 | -2 | 2 | 8 |
|---|---|---|---|
| -12 | -4 | 4 | 12 |
| -16 | -4 | 4 | 16 |
| -24 | -8 | 8 | 24 |
| -31 | -6 | 6 | 31 |
| -34 | -12 | 12 | 34 |
| -47 | -19 | 19 | 47 |
| -50 | -8 | 8 | 50 |
| -62 | -12 | 12 | 62 |
| -68 | -24 | 24 | 68 |
| -80 | -28 | 28 | 80 |
| -94 | -38 | 38 | 94 |
| -100 | -16 | 16 | 100 |
| -127 | -42 | 42 | 127 |
| -160 | -56 | 56 | 160 |
| -254 | -84 | 84 | 254 |

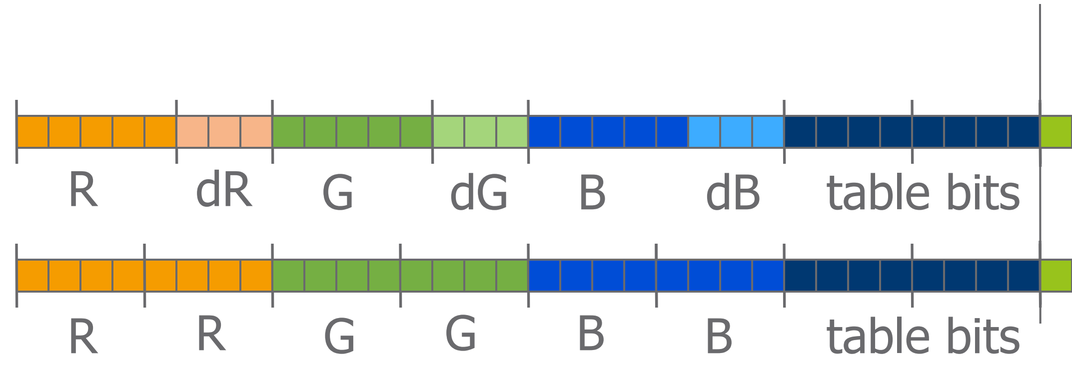
- By shrinking the codebook from 16 entries to 8, we can save one bit on each of the table code words.



# Differential Encoding

| -8 | -2 | 2 | 8 |
|---|---|---|---|
| -17 | -5 | 5 | 17 |
| -29 | -9 | 9 | 29 |
| -42 | -13 | 13 | 42 |
| -60 | -18 | 18 | 60 |
| -80 | -24 | 24 | 80 |
| -106 | -33 | 33 | 106 |
| -183 | -47 | 47 | 183 |

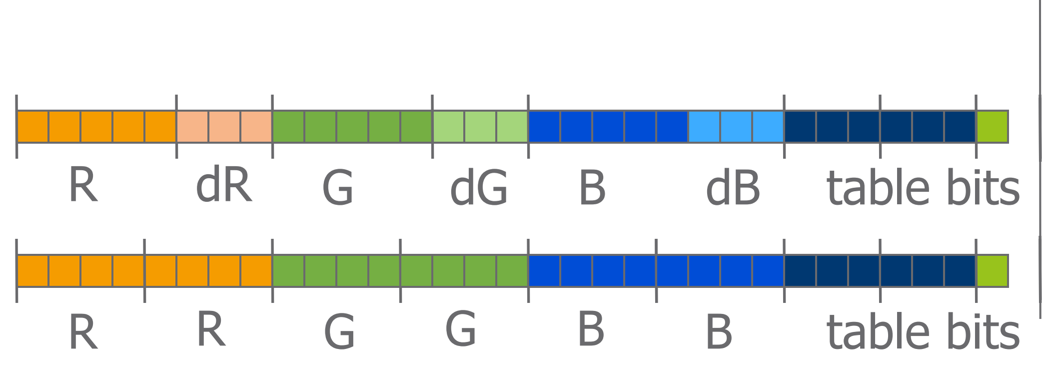
› By shrinking the codebook from 16 entries to 8, we can save one bit on each of the table code words.



# Differential Encoding

| | | | |
|---|---|---|---|
| -8 | -2 | 2 | 8 |
| -17 | -5 | 5 | 17 |
| -29 | -9 | 9 | 29 |
| -42 | -13 | 13 | 42 |
| -60 | -18 | 18 | 60 |
| -80 | -24 | 24 | 80 |
| -106 | -33 | 33 | 106 |
| -183 | -47 | 47 | 183 |

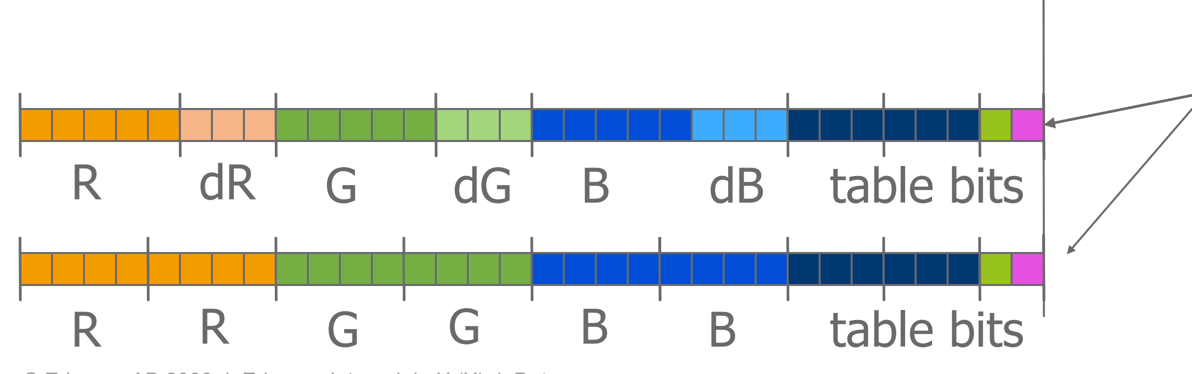
- By shrinking the codebook from 16 entries to 8, we can save one bit on each of the table code words.



# Differential Encoding

| -8 | -2 | 2 | 8 |
|---|---|---|---|
| -17 | -5 | 5 | 17 |
| -29 | -9 | 9 | 29 |
| -42 | -13 | 13 | 42 |
| -60 | -18 | 18 | 60 |
| -80 | -24 | 24 | 80 |
| -106 | -33 | 33 | 106 |
| -183 | -47 | 47 | 183 |

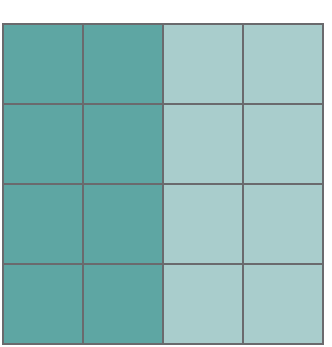
- By shrinking the codebook from 16 entries to 8, we can save one bit on each of the table code words.
- That creates room for an extra bit.

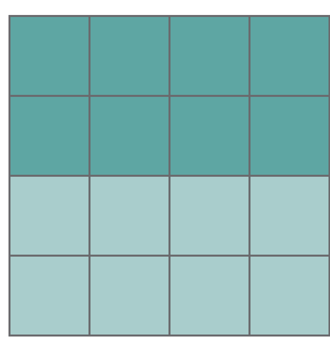



flipped or non flipped

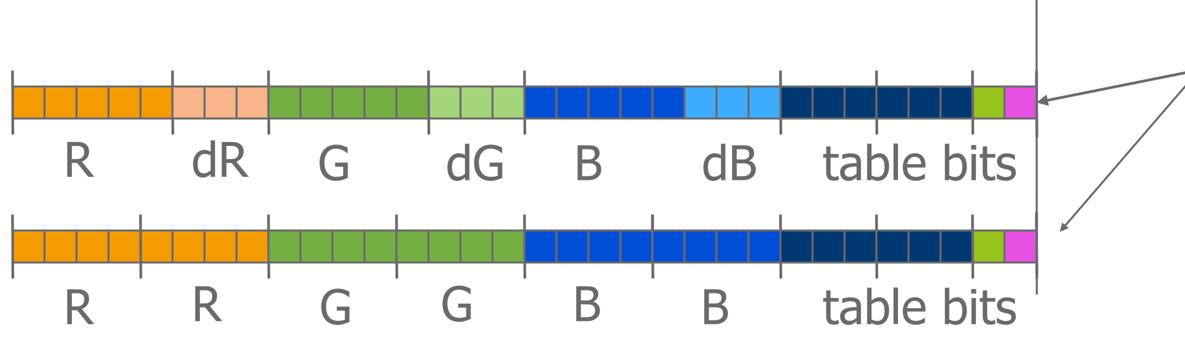
# Differential Encoding

- By shrinking the codebook from 16 entries to 8, we can save one bit on each of the table code words.
- That creates room for an extra bit.
- The new bit determines if the 2x4 blocks are vertically or horizontally oriented.





flipped or non flipped



# Results "improved PACKMAN"

## or Ericsson Texture Compression (ETC)

- Much less chrominance banding
- Jumps 2.5 dB in PSNR overall

PACKMAN ETC

# Results

# Quality Measure

- One common measure is the Root Mean Square Error (RMSE) measure.

$$RMSE = \sqrt{\frac{1}{w \times h} \sum_{x,y} \Delta R_{xy}^2 + \Delta G_{xy}^2 + \Delta B_{xy}^2},$$

# Quality Measure

- One common measure is the Root Mean Square Error (RMSE) measure.

$$RMSE = \sqrt{\frac{1}{w \times h} \sum_{x,y} \Delta R_{xy}^2 + \Delta G_{xy}^2 + \Delta B_{xy}^2},$$

- A variant of this is Peak Signal to Noise ratio (PSNR)

$$PSNR = 10 \log_{10} \left( \frac{3 \times 255^2}{RMSE^2} \right),$$

# Quality Measure

- One common measure is the Root Mean Square Error (RMSE) measure.

$$RMSE = \sqrt{\frac{1}{w \times h} \sum_{x,y} \Delta R_{xy}^2 + \Delta G_{xy}^2 + \Delta B_{xy}^2},$$

- A variant of this is Peak Signal to Noise ratio (PSNR)

$$PSNR = 10 \log_{10} \left( \frac{3 \times 255^2}{RMSE^2} \right),$$

- Usually, 0.25 dB difference is a visible difference.

# Results

## We have compared against the following systems:

| Scheme | Coder Used |
| --- | --- |
| S3TC/DXTC [Iourcha et al. ‘99] | ATI’s Compressonator, with weights [1, 1, 1] to maximize PSNR |

# Results

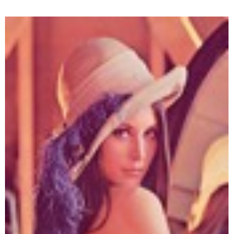
We have compared against the following systems:

| Scheme | Coder Used |
| --- | --- |
| S3TC/DXTC [Iourcha et al. '99] | ATI's Compressonator, with weights [1, 1, 1] to maximize PSNR |
| PVR-TC [Fenney '03] | No coder publicly available – the same images were used and results taken from the paper. |

We have compared against the following systems:

| Scheme | Coder Used |
|---|---|
| S3TC/DXTC [Iourcha et al. '99] | ATI's Compressonator, with weights [1, 1, 1] to maximize PSNR |
| PVR-TC [Fenney '03] | No coder publicly available – the same images were used and results taken from the paper. |

The 7 images used were:

Lena  Lorikeet Kodim1-5

# Results

We have compared against the following systems:

| Scheme | Coder Used |
| --- | --- |
| S3TC/DXTC [Iourcha et al. '99] | ATI's Compressonator, with weights [1, 1, 1] to maximize PSNR |
| PVR-TC [Fenney '03] | No coder publicly available – the same images were used and results taken from the paper. |
| PACKMAN [Strom and Akenine-Moller '04] | Exhaustive Coding |

# Results for ETC

We have compared against the following systems:

| Scheme | Coder Used | Average Gain |
| --- | --- | --- |
| S3TC/DXTC [Iourcha et al. ‘99] | ATI’s Compressonator, with weights [1, 1, 1] to maximize PSNR | 0.41 dB |
| PVR-TC [Fenney ’03] | No coder publicly available – the same images were used and results taken from the paper. | 0.65 dB |
| PACKMAN [Strom and Akenine-Moller ’04] | Exhaustive Coding | 2.5 dB |

# Results for ETC

We have compared against the following systems:

| Scheme | Coder Used | Average Gain |
|---|---|---|
| S3TC/DXTC [Iourcha et al. '99] | ATI's Compressonator, with weights [1, 1, 1] to maximize PSNR | 0.41 dB |
| PVR-TC [Fenney '03] | No coder publicly available – the same images were used and results taken from the paper. | 0.65 dB |
| PACKMAN [Strom and Akenine-Moller '04] | Exhaustive Coding | 2.5 dB |

These figures were collected from a rather small number of images. When using more images, quality of ETC was similar to S3TC/DXTC.

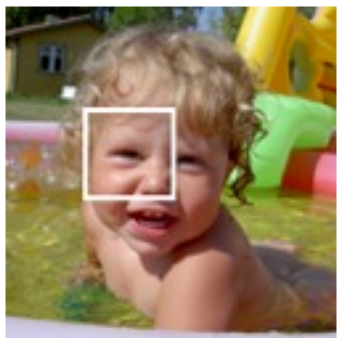
# Strengths

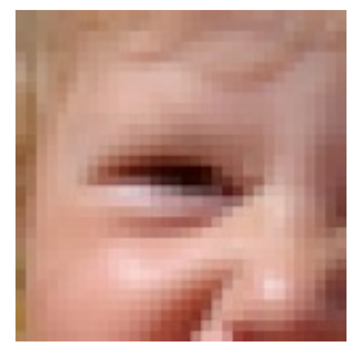
- The strengths can most easily be seen in areas with fine variations in luminance.

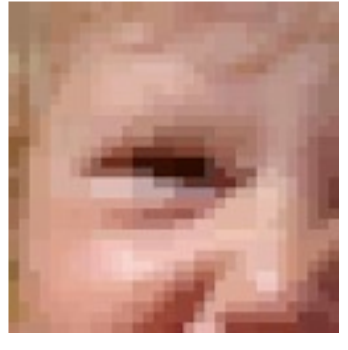
original

# Strengths

- The strengths can most easily be seen in areas with fine variations in luminance.



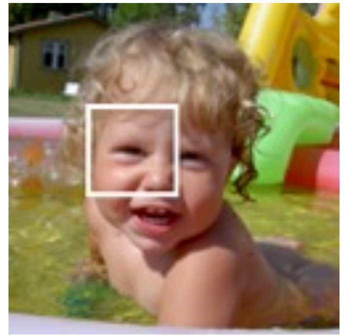


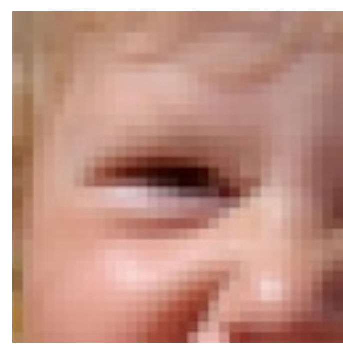


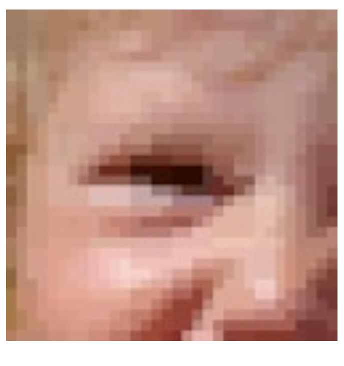
S3TC/DXTC

# Strengths

- The strengths can most easily be seen in areas with fine variations in luminance.







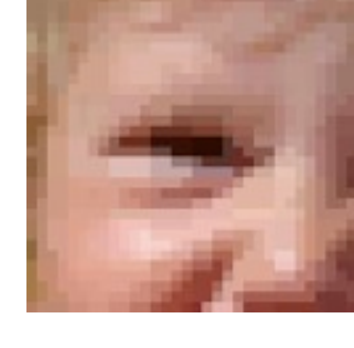
S3TC/DXTC

ETC

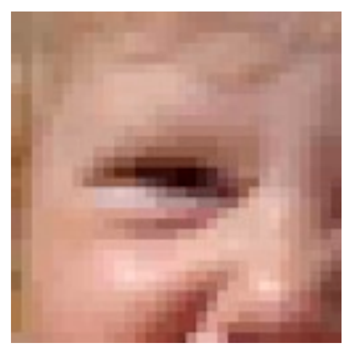
# Strengths

- The strengths can most easily be seen in areas with fine variations in luminance.

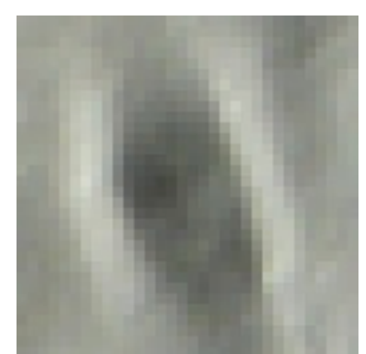
original



S3TC/DXTC

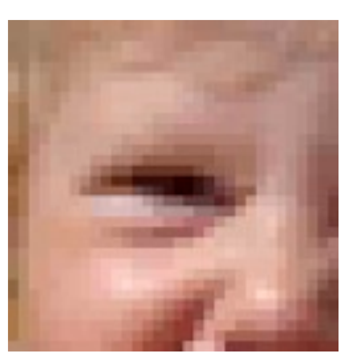


ETC

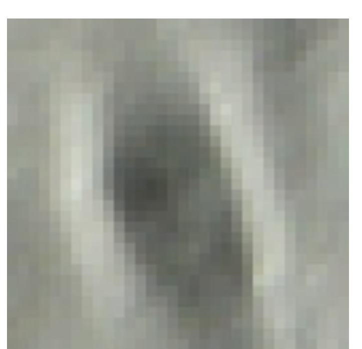


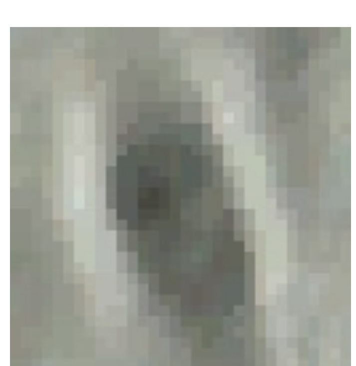
# Strengths

- The strengths can most easily be seen in areas with fine variations in luminance.



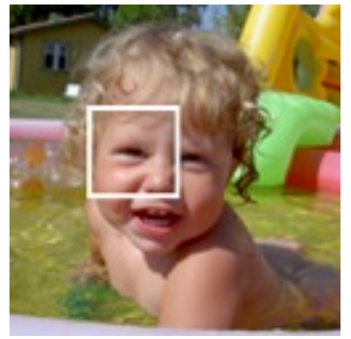
original S3TC/DXTC ETC



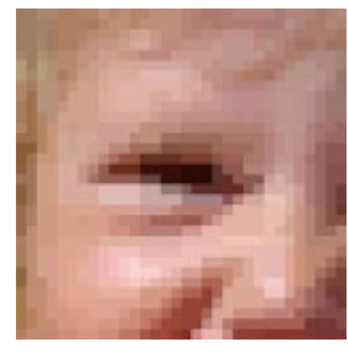


# Strengths

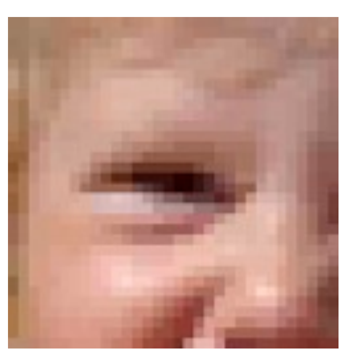
- The strengths can most easily be seen in areas with fine variations in luminance.



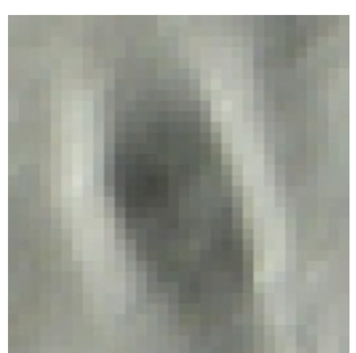
original

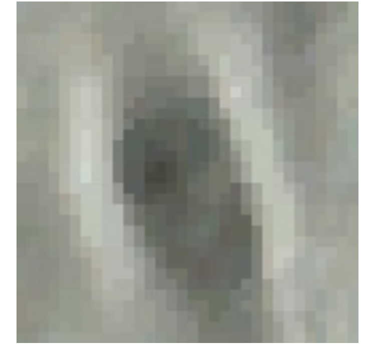


S3TC/DXTC



ETC



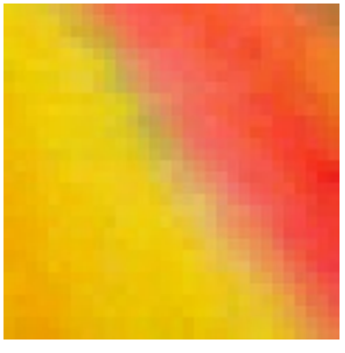


# Weaknesses

- When there are more than two colors of different chrominance in a 2x4 block, ETC has problems.
- Such artifacts are especially visible when the two colors have similar luminance.

# Weaknesses

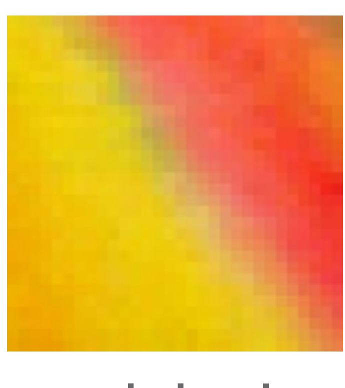
- When there are more than two colors of different chrominance in a 2x4 block, ETC has problems.
- Such artifacts are especially visible when the two colors have similar luminance.



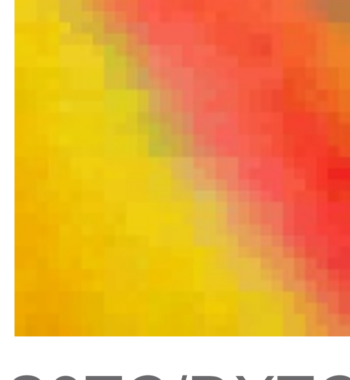
original

# Weaknesses

- When there are more than two colors of different chrominance in a 2x4 block, ETC has problems.
- Such artifacts are especially visible when the two colors have similar luminance.



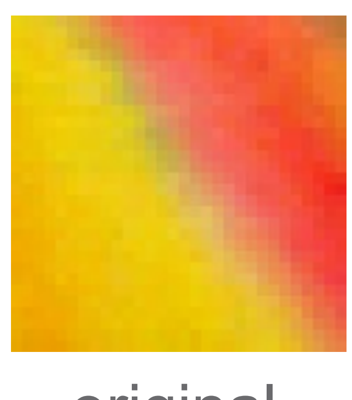
original



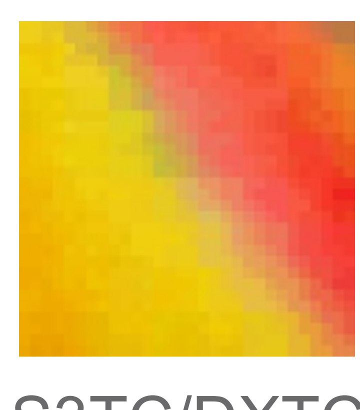
S3TC/DXTC

# Weaknesses

- When there are more than two colors of different chrominance in a 2x4 block, ETC has problems.
- Such artifacts are especially visible when the two colors have similar luminance.



original



S3TC/DXTC

ETC

# Adoption

- The Khronos organization has adopted ETC under the name "Ericsson Texture Compression" (ETC) through an OES extension for OpenGL ES.
- It is likely to be used by M3G 2.0, the new Java standard for 3D graphics on phones
- Independent hardware vendors have started implementing ETC.

# ETC2

- Recently, an updated version of ETC was presented:
  - J. Ström and M. Pettersson ”ETC2: Texture Compression Using Invalid Combinations”, *Graphics Hardware* 2007
- It is backwards compatible to ETC and brings a 1.0 dB increase in quality compared to ETC. (0.8 dB compared to S3TC/DXTC)
- It fixes mostly blocks that ETC has problems with

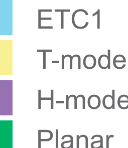
# Results

original S3TC/DXTC ETC1 ETC2

ETC1
T-mode
H-mode
Planar

original S3TC/DXTC ETC 1 ETC2

# Results

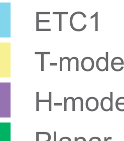



original | S3TC/DXTC | ETC1 | ETC2

original | S3TC/DXTC | ETC 1 | ETC2

© Ericsson AB 2009 | Ericsson Internal | X (X) | Date

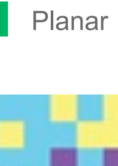
# Results




original | S3TC/DXTC | ETC1 | ETC2

original | S3TC/DXTC | ETC 1 | ETC2

# Results
cont.




original S3TC/DXTC ETC1 ETC2

original S3TC/DXTC ETC1 ETC2

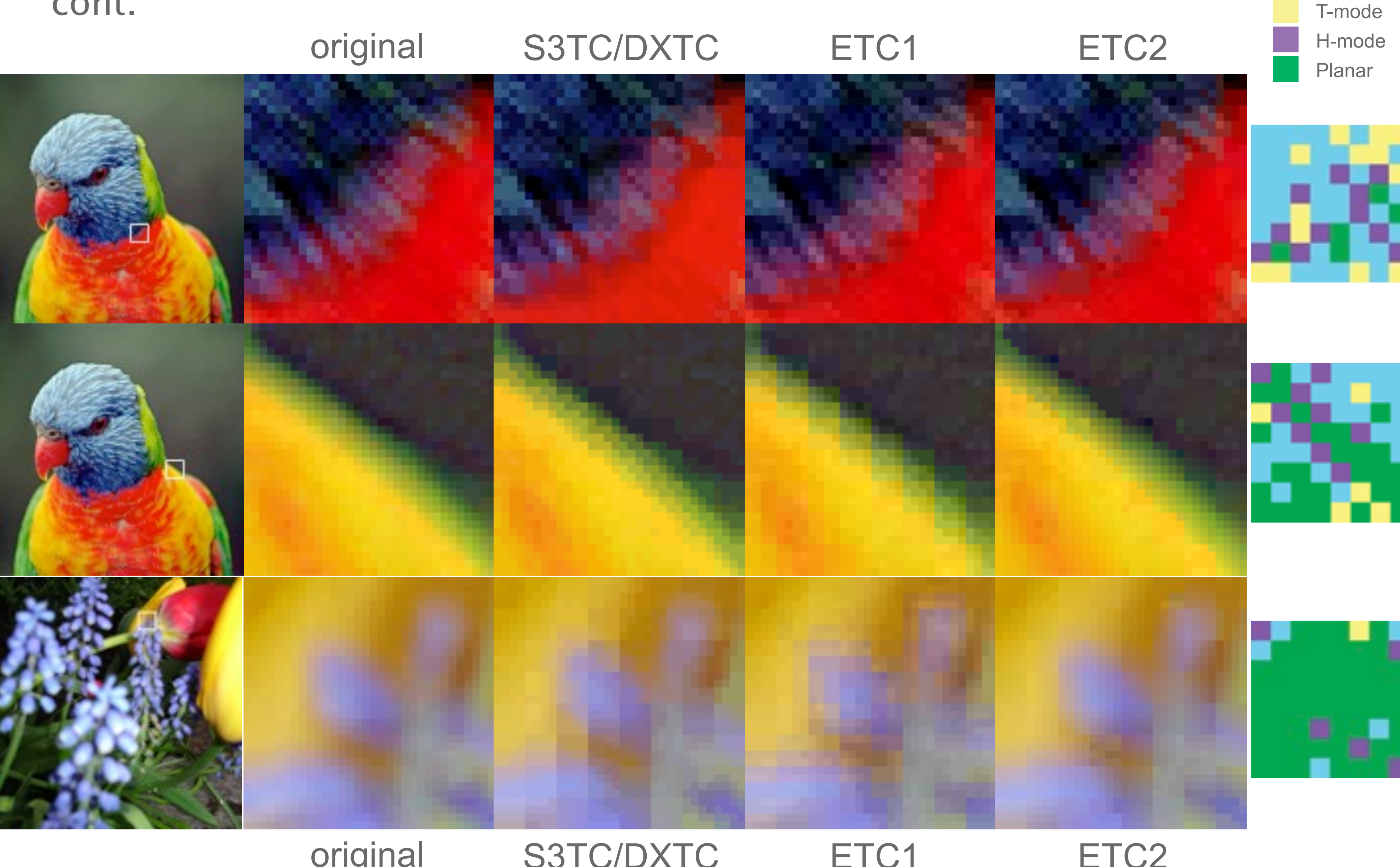
# Results
cont.

original S3TC/DXTC ETC1 ETC2

ETC1
T-mode
H-mode
Planar

original S3TC/DXTC ETC1 ETC2

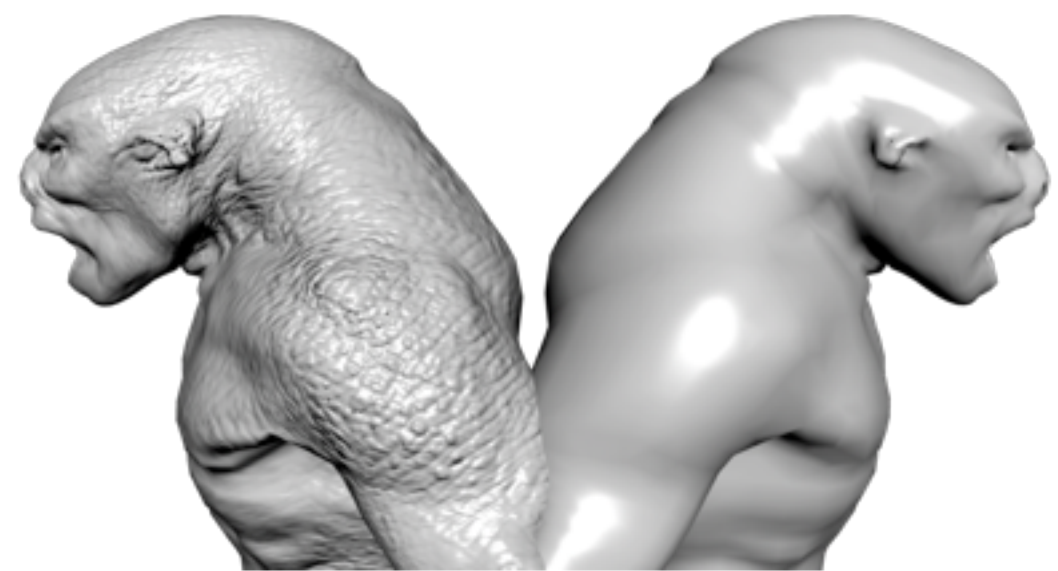
# Results
cont.



# Normal Map Compression

## Overview

- Normal maps: definition and usage
- The 3Dc algorithm
- Improvements over 3Dc

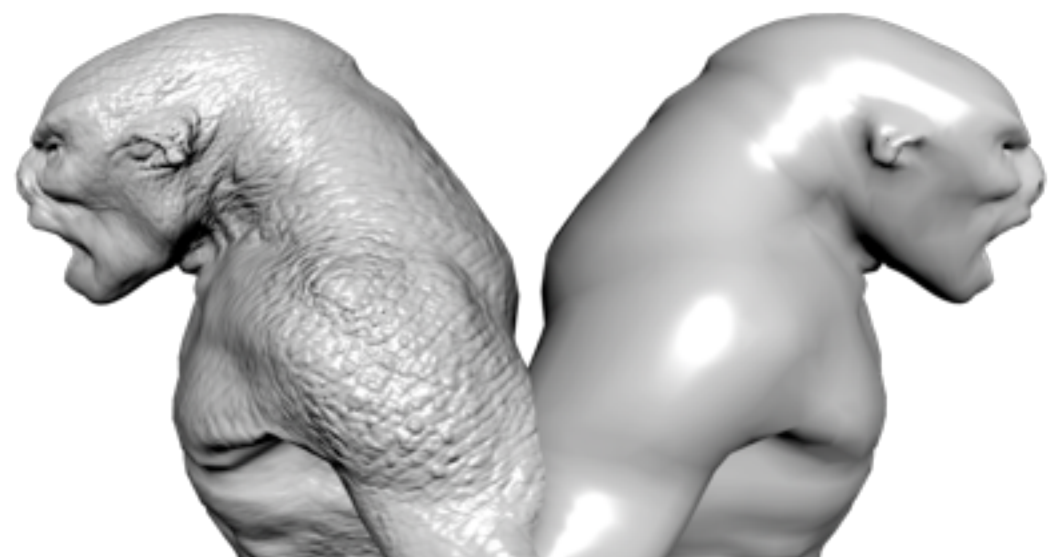


# Normal Map Compression

## Overview

- Normal maps: definition and usage
- The 3Dc algorithm
- Improvements over 3Dc

I've got normal maps

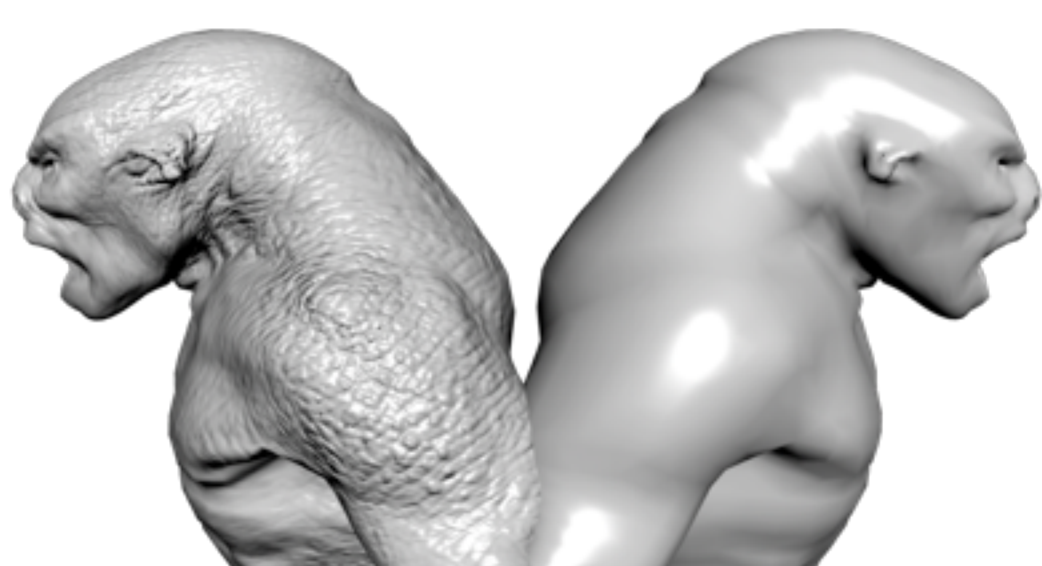


# Normal Map Compression

## Overview

- Normal maps: definition and usage
- The 3Dc algorithm
- Improvements over 3Dc

I’ve got normal maps



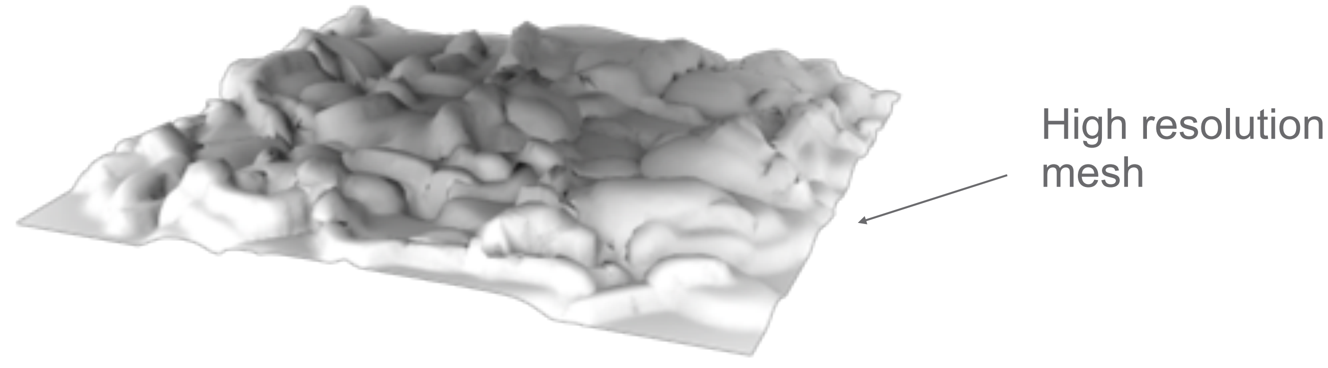
I’ve got nothing

# Normal Maps Usage

- Adds geometric detail with the help of texture maps
- Stores a value of the local normal vector
- Realistic, detailed appearance at low cost

# How to ”bake” a Normal Map

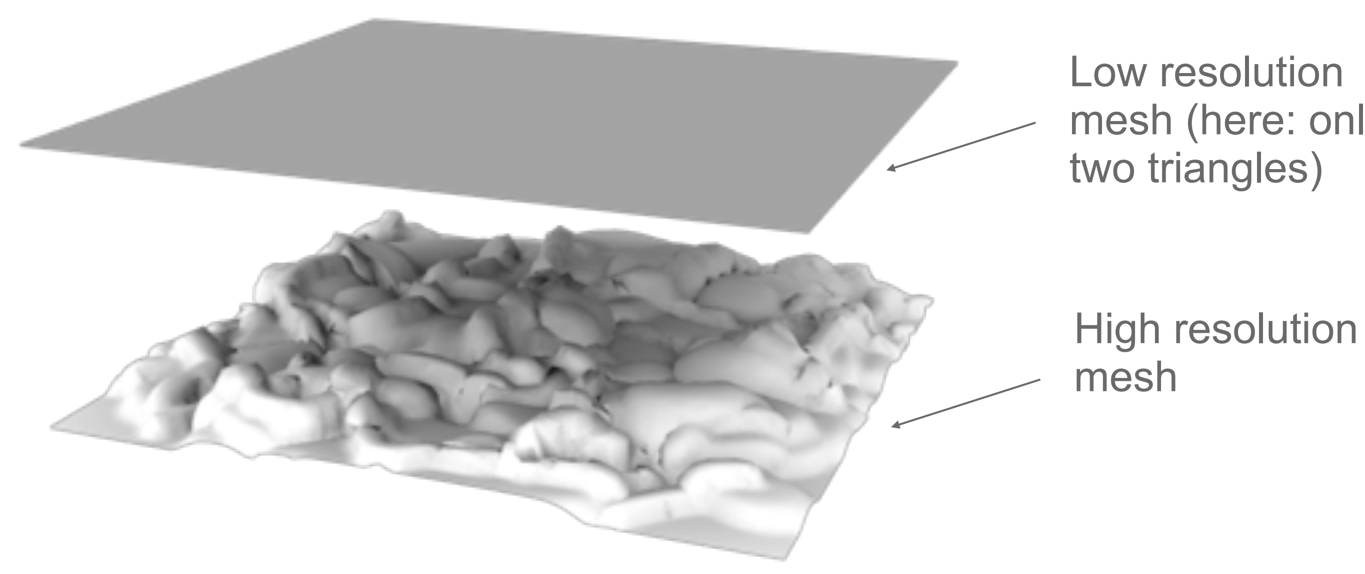
- Use a original high resolution mesh



High resolution mesh

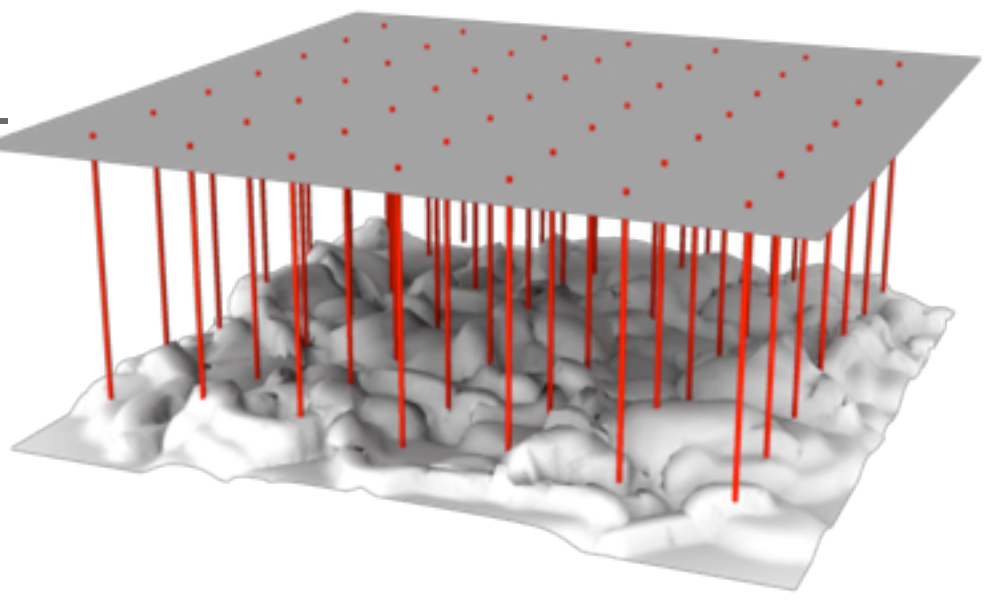
# How to ”bake” a Normal Map

- Use a original high resolution mesh
- Create a low-res mesh that captures overall shape



# How to ”bake” a Normal Map
(continued)

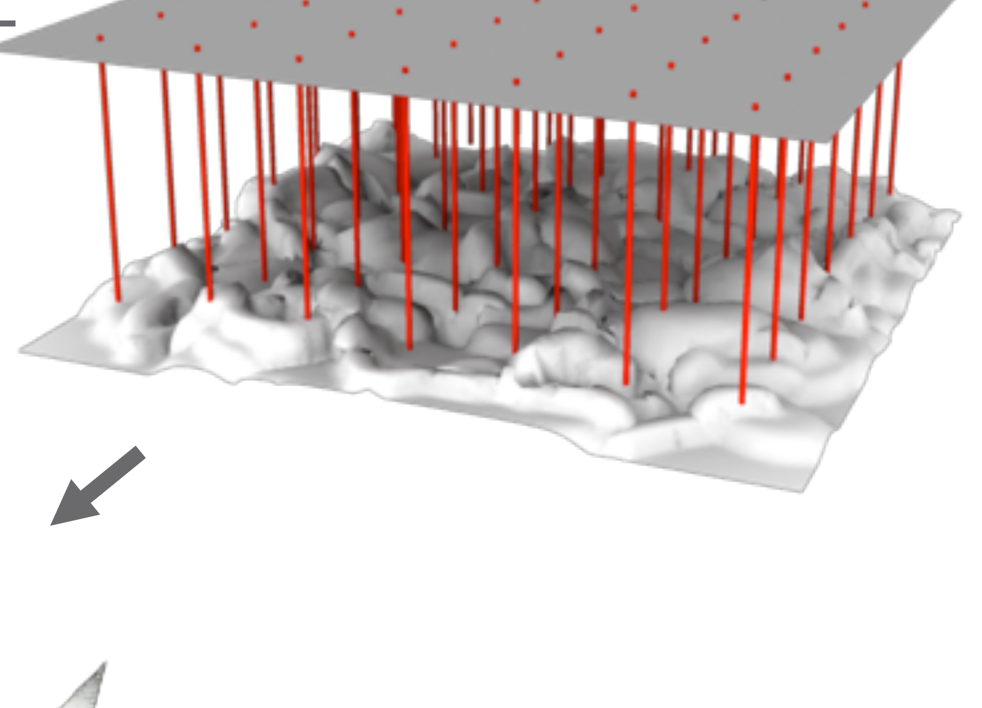
- Shoot rays from the lo-res surface to the high-res surface



# How to ”bake” a Normal Map
(continued)

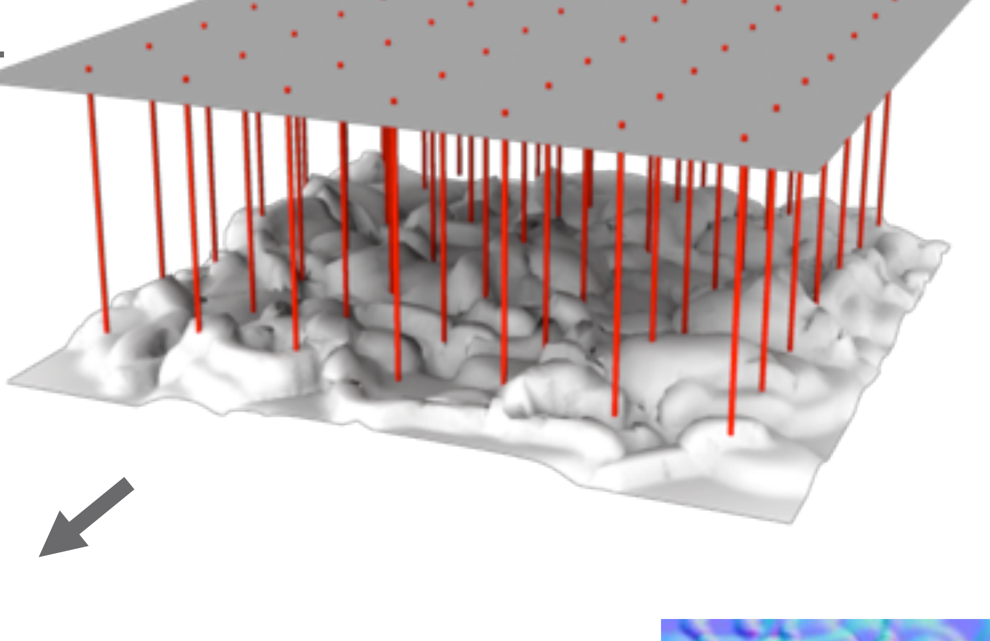
- Shoot rays from the lo-res surface to the high-res surface

- Calculate the normal vector (X, Y, Z) in the intersection points...



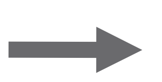
# How to "bake" a Normal Map
(continued)

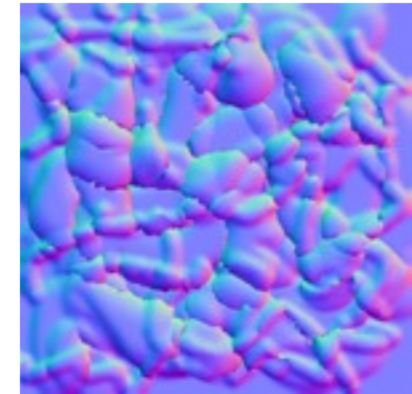
- Shoot rays from the lo-res surface to the high-res surface



- Calculate the normal vector (X, Y, Z) in the intersection points...

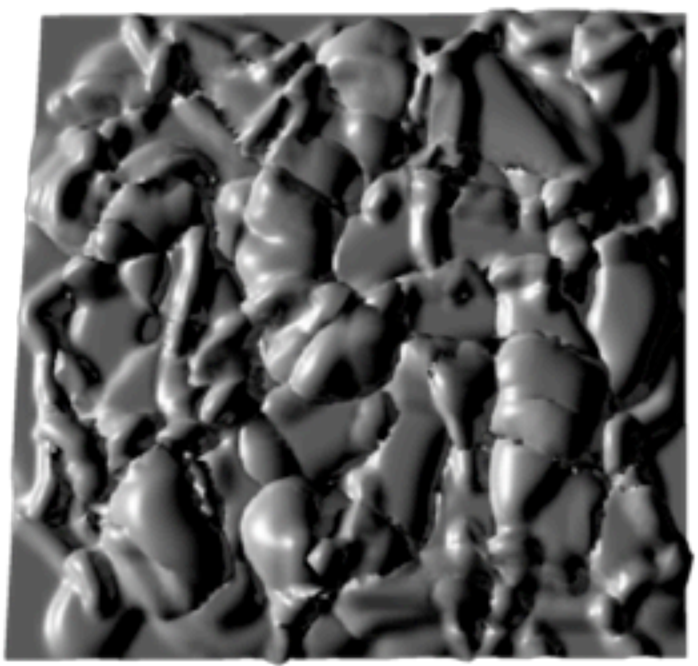
... and store them as RGB in a texture



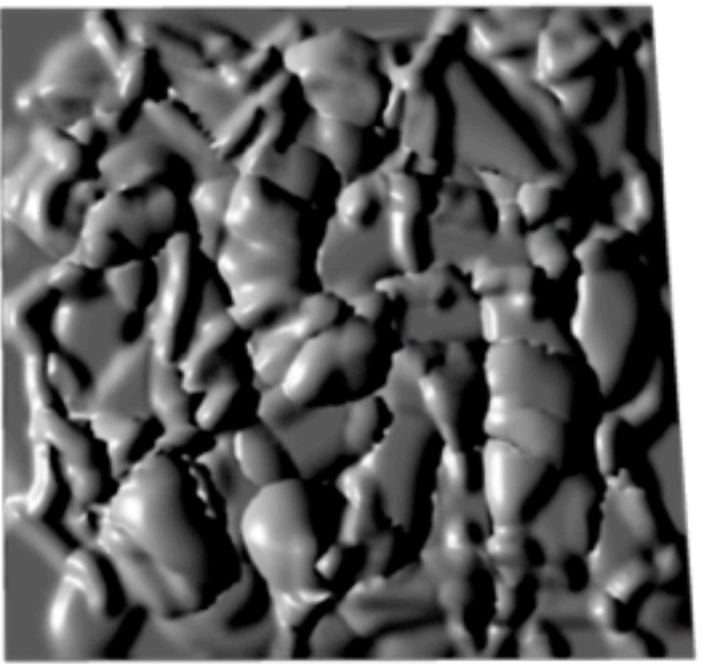


# How to ”bake” a Normal Map
(continued)

› Render the low-res surface + normal map



Hi-res – 20k triangles



Lo-res – two triangles + normal map

# We need compression!

› Render the low-res surface + normal map

Hi-res – 20k triangles

Lo-res – two triangles + normal map

Increased Memory bandwidth!!!

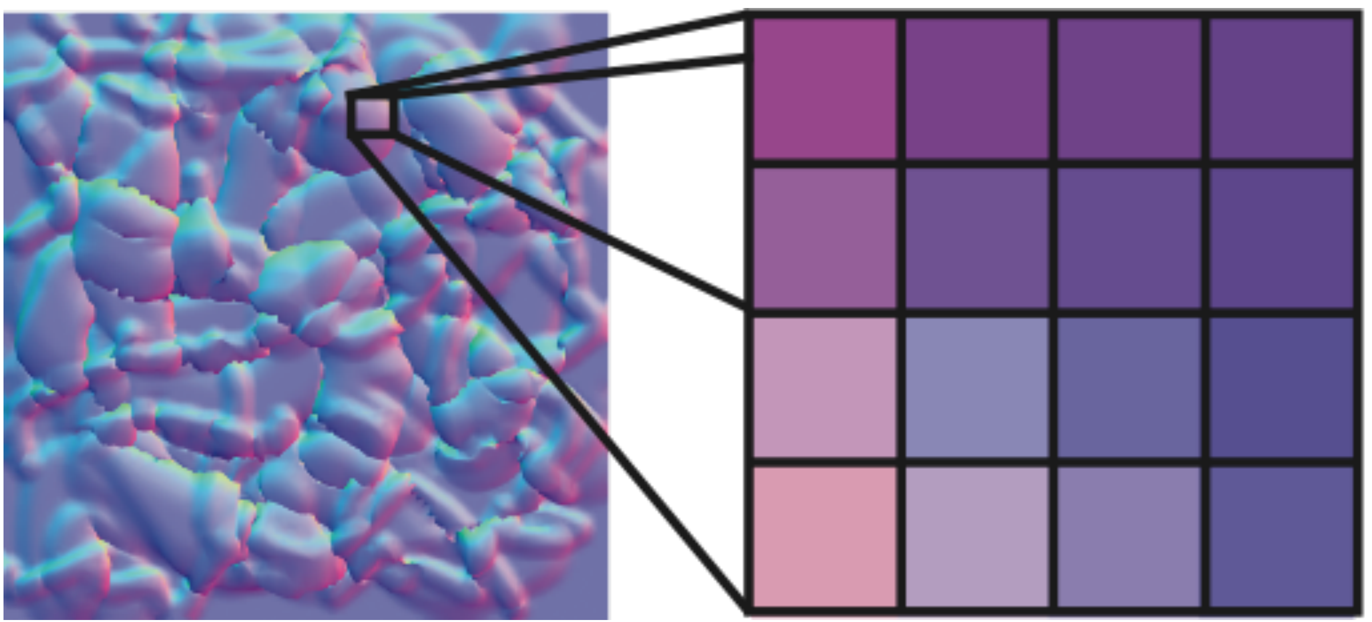
# Why not use regular texture comp.?

- S3TC and ETC are designed for colors – not for normal data
- Visible artifacts along edges and in smooth areas
- It seems more than 4 bits per pixel is usually needed

- There are two methods especially made for normal maps:
  - 3Dc by ATI – 8 bits per pixel
  - Ericsson Normal Compression, ENC (mostly developed by Lund University [Munkberg et al. 2006]), also 8 bits per pixel

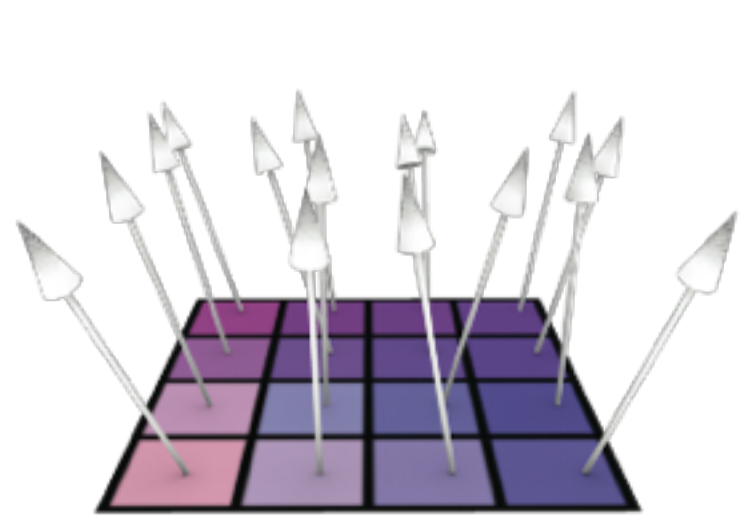
# 3Dc Overview

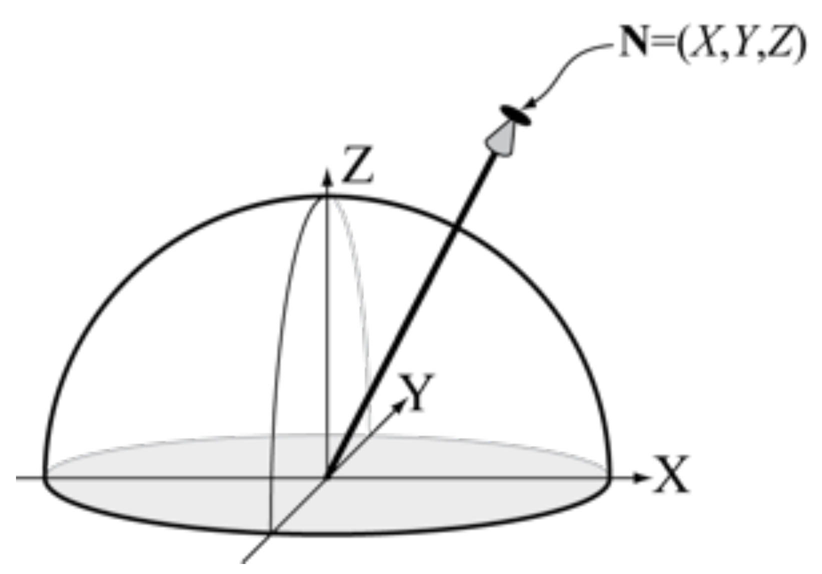
- Divide the normal map texture into 4x4 texel blocks



# 3Dc Overview
## Continued

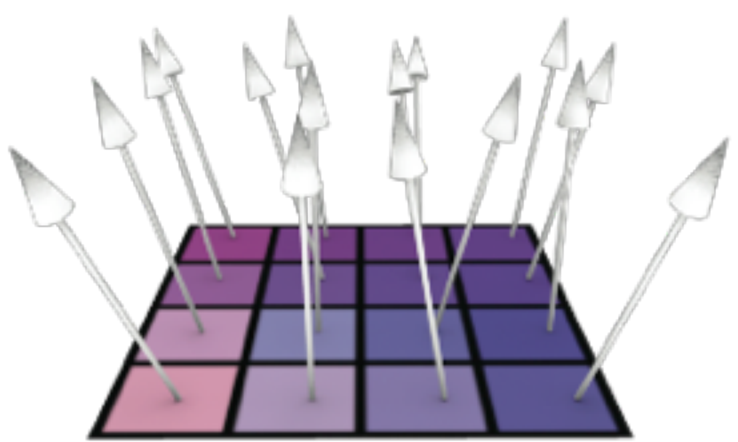
- Each normal vector is normalized to the unit sphere
  - $(X, Y, Z) \quad \underset{\longrightarrow}{(x, y, z)}$

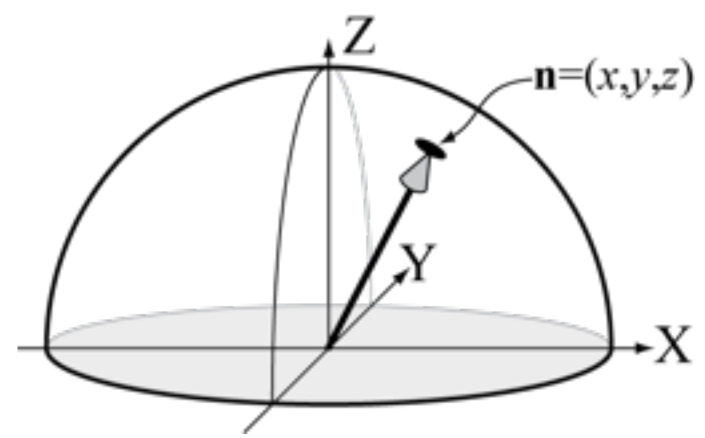




# 3Dc Overview
## Continued

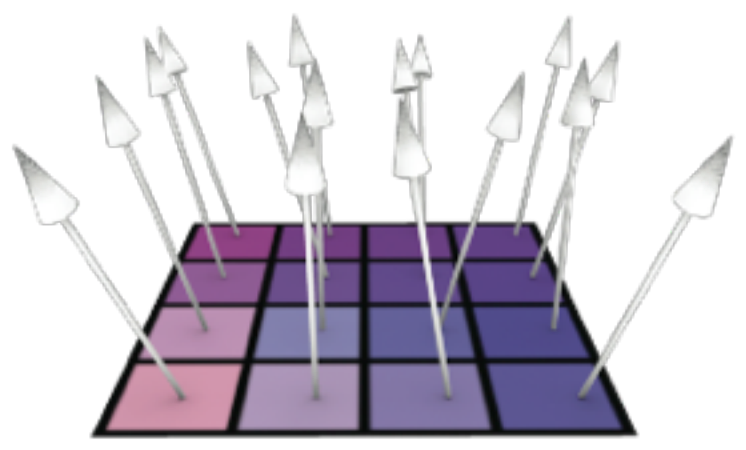
- Each normal vector is normalized to the unit sphere
  - $(X, Y, Z) \longrightarrow (x, y, z)$

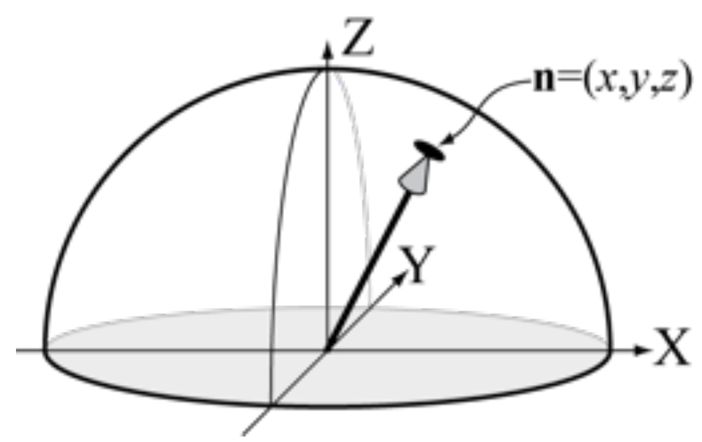




# 3Dc Overview
## Continued

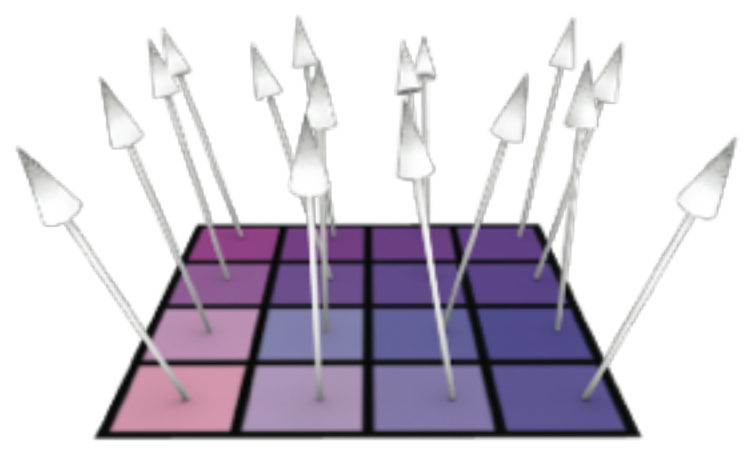
- Each normal vector is normalized to the unit sphere
  - $(X, Y, Z) \quad \overrightarrow{\quad} \quad (x, y, z)$
- The $z$-coordinate can then be calculated using $z = \sqrt{1 - x^2 - y^2}$ and does not need to be stored.

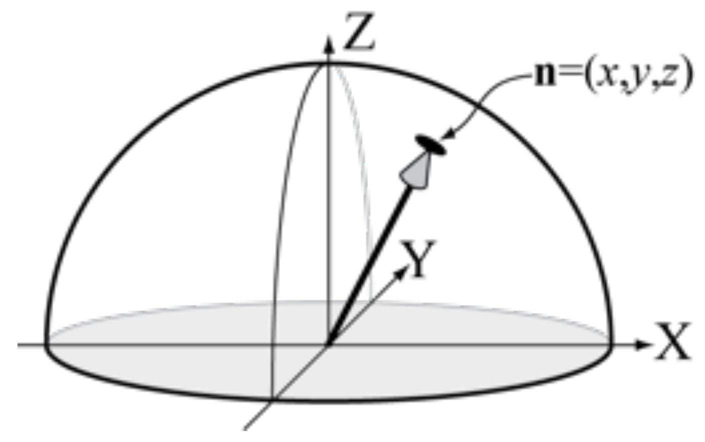




# 3Dc Overview
## Continued

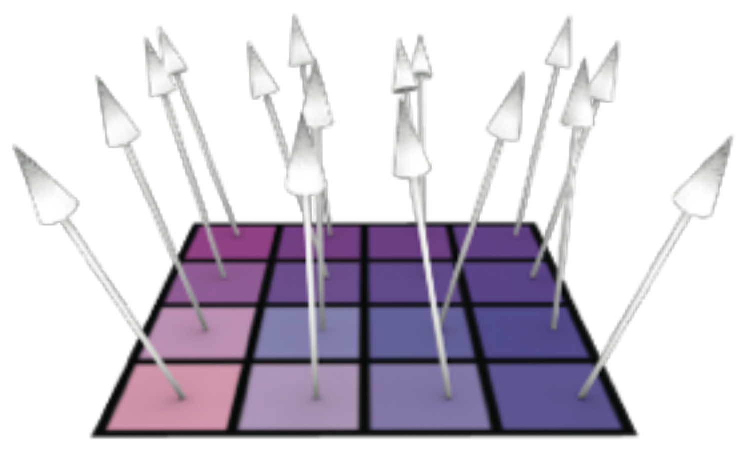
- Each normal vector is normalized to the unit sphere
  - $(X, Y, Z) \rightarrow (x, y, z)$
- The $z$-coordinate can then be calculated using $z = \sqrt{1 - x^2 - y^2}$ and does not need to be stored.
- Only the $(x,y)$ projection of the vector is stored.

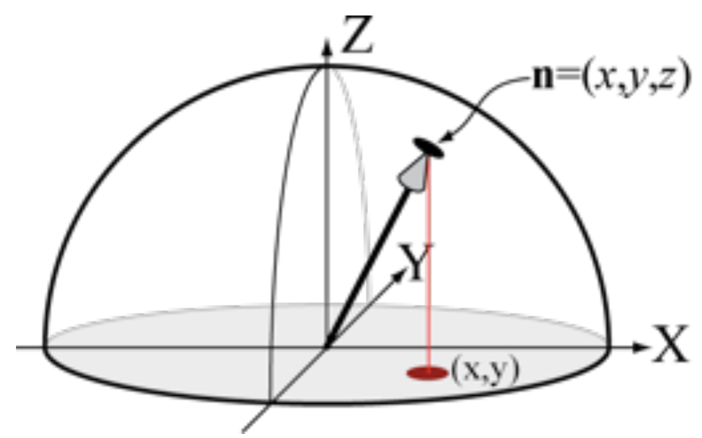




# 3Dc Overview
## Continued

- Each normal vector is normalized to the unit sphere
  - $(X, Y, Z) \rightarrow (x, y, z)$
- The $z$-coordinate can then be calculated using $z = \sqrt{1 - x^2 - y^2}$ and does not need to be stored.
- Only the $(x,y)$ projection of the vector is stored.





# 3Dc Overview
(Continued)

› The bounding box for the block's projected normals in the $XY$-plane is stored

xmin xmax

Z

n=(x,y,z)

Y

X

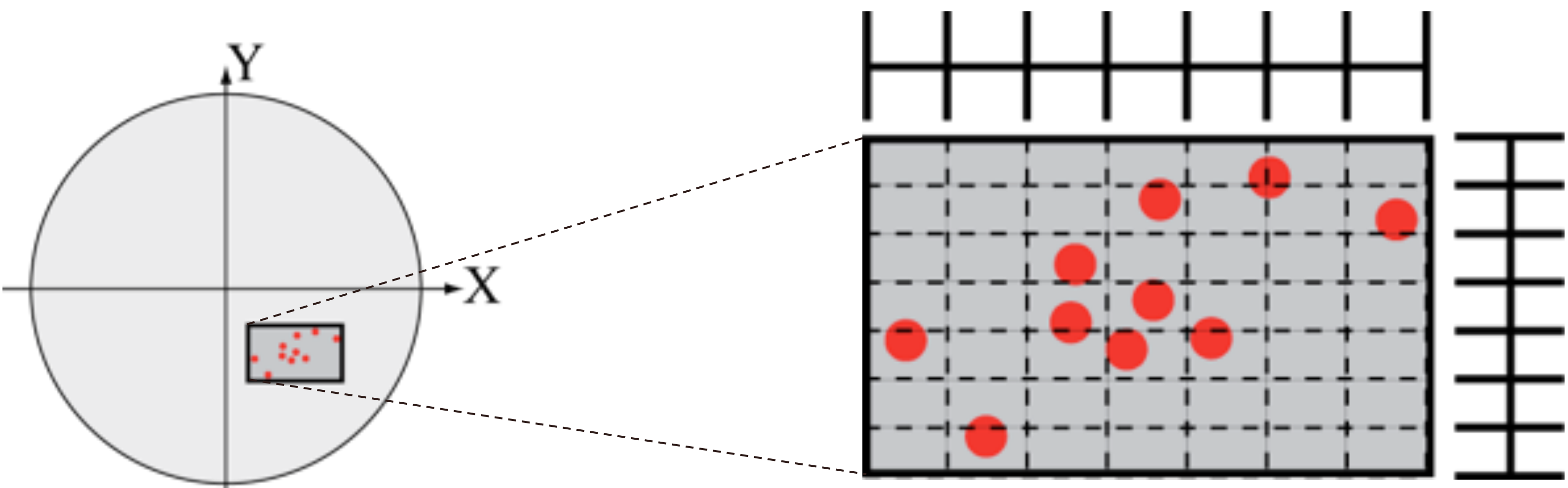
(x,y)

ymin

ymax

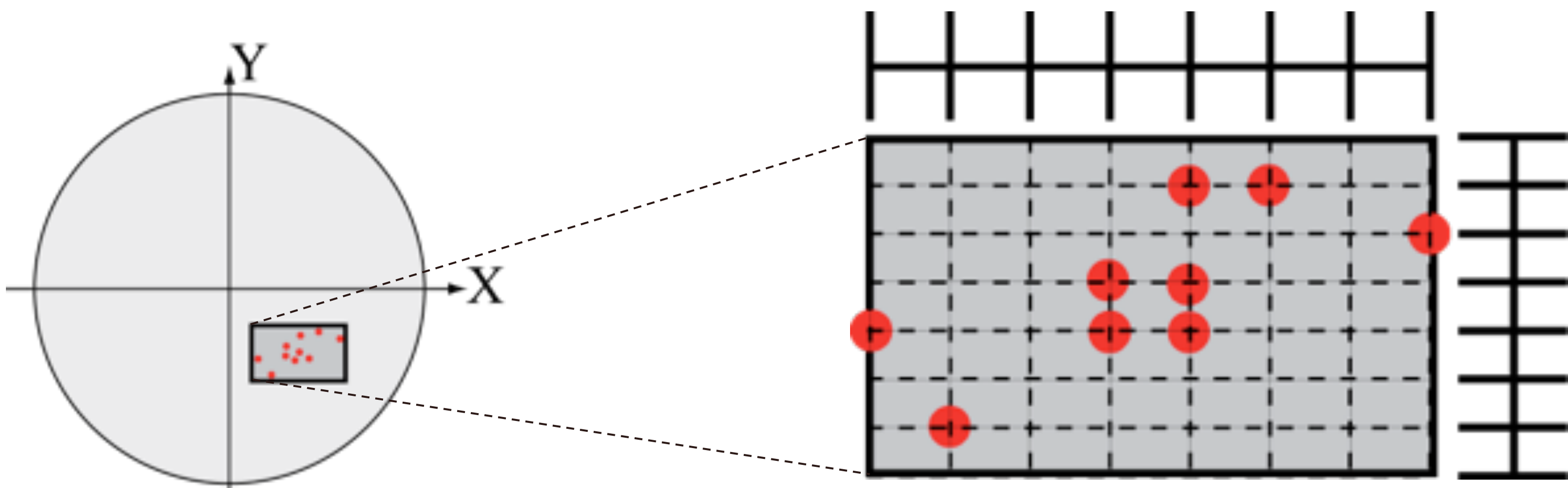
# 3Dc Overview
(Continued)

› Inside the bounding box, each dimension is quantized to one of eight levels (3 bits per dimension)



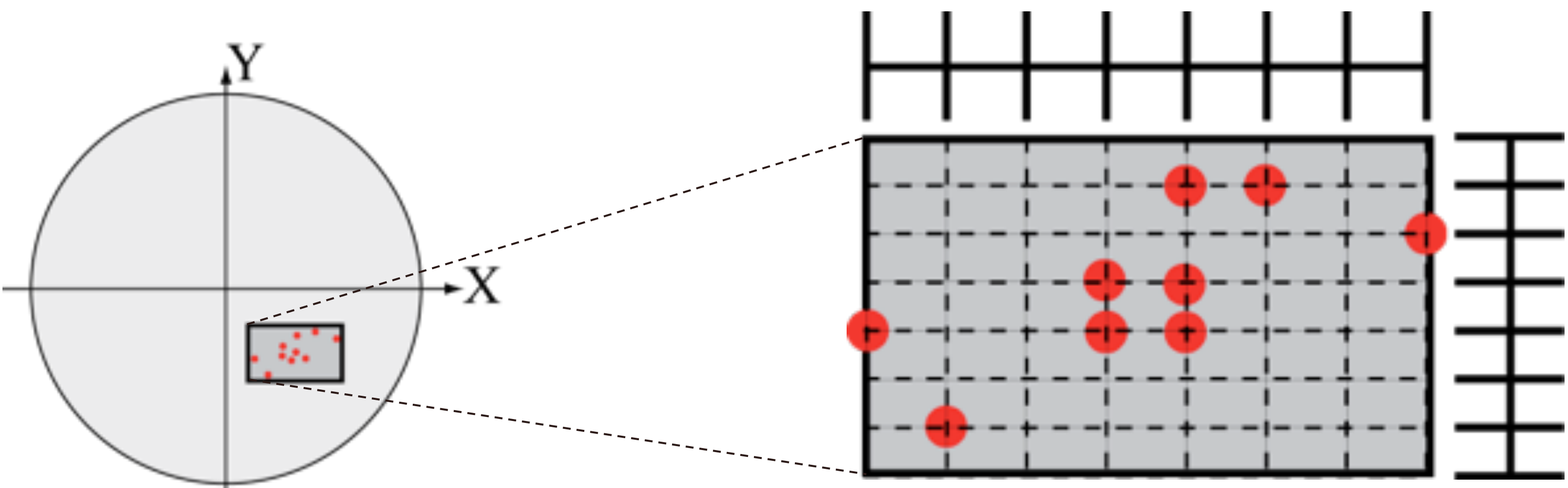
# 3Dc Overview
(Continued)

- Inside the bounding box, each dimension is quantized to one of eight levels (3 bits per dimension)



# 3Dc Overview
(Continued)

- Inside the bounding box, each dimension is quantized to one of eight levels (3 bits per dimension)
- In total 128 bits per block:
  - 32 bits for bounding box (8bits per xmin,xmax,ymin,ymax)
  - 6 texel index bits per pixel = 96 bits



# 3Dc Decompression

- Restore x and y values from the min/max values and the texel indices using

# 3Dc Decompression

- Restore x and y values from the min/max values and the texel indices using
  - $x$ = xmin + index * (xmax-xmin)/7

# 3Dc Decompression

- Restore x and y values from the min/max values and the texel indices using
  - $x$ = xmin + index * (xmax-xmin)/7
  - $y$ = ymin + index * (ymax-ymin)/7

# 3Dc Decompression

- Restore x and y values from the min/max values and the texel indices using
  - $x$ = xmin + index * (xmax-xmin)/7
  - $y$ = ymin + index * (ymax-ymin)/7
- Restore unit z-values using

$$z = \sqrt{1 - x^2 - y^2}$$

# 3Dc Decompression

- Restore x and y values from the min/max values and the texel indices using
  - $x$ = xmin + index * (xmax-xmin)/7
  - $y$ = ymin + index * (ymax-ymin)/7
- Restore unit z-values using
- Can be done in a pixel shader

$$z = \sqrt{1 - x^2 - y^2}$$

# 3Dc Decompression

- Restore x and y values from the min/max values and the texel indices using
  - $x$ = xmin + index * (xmax-xmin)/7
  - $y$ = ymin + index * (ymax-ymin)/7
- Restore unit z-values using
- Can be done in a pixel shader

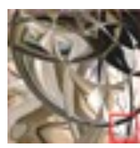
$$z = \sqrt{1 - x^2 - y^2}$$

- Supported by AMD graphics cards

# Newer Normal Map Compression

## Techniques

- Jacob Munkberg, Tomas Akenine-Möller and Jacob Ström,"High-Quality Normal Map Compression", *Graphics Hardware*, September 2006.
  - 1.87 dB higher quality than 3Dc
  - Backwards compatible with 3Dc



- Jacob Munkberg, Ola Olsson, Jacob Ström and Tomas Akenine-Möller, "Tight Frame Normal Map Compression" *Graphics Hardware*, 2007
  - 2.63 dB higher quality than 3Dc
  - A candidate for inclusion in OpenGL ES

# Thanks to:

- Jacob Munkberg (for Normal Map Compression slides)
- Michael Doggett (for inviting me)
- You (for listening!)

# Thank you

ERICSSON