Image Compression
Part 2: Example of DXTC
and 3Dc
Example of DXTC

- Suppose we are given a color texture represented in R8G8B8 format.

\[(R,G,B) = (192,150,128)\]
Example of DXTC

- Divide image into 4x4 blocks
Example of DXTC

- Store two 16-bit representative color values C0 (high) and C1 (low) in R5G6B5 format

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(188, 146, 124)</td>
<td>(188, 146, 124)</td>
<td>(187, 145, 123)</td>
<td>(183, 143, 118)</td>
</tr>
<tr>
<td>(186, 144, 122)</td>
<td>(187, 142, 121)</td>
<td>(187, 142, 121)</td>
<td>(184, 144, 119)</td>
</tr>
<tr>
<td>(186, 144, 122)</td>
<td>(187, 142, 121)</td>
<td>(187, 142, 121)</td>
<td>(182, 142, 117)</td>
</tr>
<tr>
<td>(186, 144, 122)</td>
<td>(187, 142, 121)</td>
<td>(187, 142, 121)</td>
<td>(184, 144, 119)</td>
</tr>
</tbody>
</table>

(R,G,B)=(188,146,124)

(R,G,B)=(24,37,16)  
(R,G,B)=(182,142,117)  
(R,G,B)=(23,36,15)
Example of DXTC

- Compute two additional color values
- (e.g., using simple interpolation)

\[
\begin{align*}
C_0 & : (R,G,B) = (24,37,16) \\
C_2 & : (R,G,B) = (23.67,36.67,15.67) \\
C_3 & : (R,G,B) = (23.33,36.33,15.333) \\
C_1 & : (R,G,B) = (23,36,15)
\end{align*}
\]
Assign a value from 0 to 3 to each pixel based on closest color value

(R,G,B)=(24,37,16)  C0

(R,G,B)=(23.67,36.67,15.67)  C2

(R,G,B)=(23.33,36.33,15.333)  C3

(R,G,B)=(23,36,15)  C1
Example of DXTC

To decode, replace values from lookup table with one of the four color values

\[(R, G, B) = (24, 37, 16)\]  C0

\[(R, G, B) = (23.67, 36.67, 15.67)\]  C2

\[(R, G, B) = (23.33, 36.33, 15.333)\]  C3

\[(R, G, B) = (23, 36, 15)\]  C1

(23.67, 36.67, 15.67)

(189, 146, 125)
Normal Mapping

• Complex 3D models in a scene provide a greater sense of realism within a 3D environment
• However, it is expensive from both a computational and memory perspective to process such complex 3D models with high geometric detail
• Solution: use normal mapping to give the sense that there is more geometric detail by changing lighting based on supposed geometry
Normal Mapping
Creating Normal Maps

- Create high resolution model and a corresponding low resolution model you want to use
- Cast ray from each texel on low-res model
- Find intersection of ray with high-res model
- Save the normal from high-res model where the ray intersects
Normal Mapping
3Dc

- Each pixel in a normal map has three values (x,y,z), which represent a normal vector.
- The x,y, and z coordinates of a normal vector are independent from each other.
- This makes DXTC poorly suited for compressing normal maps since it relies on inter-channel correlations.
- Solution: 3Dc, an extension of BTC for normal maps.
How does 3Dc work?

- Instead of operating on all channels together, treat x, y, and z coordinate channels separate from each other.
- In most systems, all normal vectors are unit vectors with a length of 1.
- Also, z component assumed to be positive since it should point out of the surface.
How does 3Dc work?

- Idea: Instead of storing z, compute z based on x and y

\[ z = \sqrt{1 - (x^2 + y^2)} \]

- Since z is not stored, storage requirements have effectively been reduced by 1/3!
How does 3Dc encoding work?

- **Steps:**
  - Discard z channel
  - For the x and y channels, divide normal map into 4x4 blocks
  - For each block, store two 8-bit representative coordinate values (V0 and V1)
  - Compute 6 intermediate coordinate values by using simple linear interpolation between V0 and V1
How does 3Dc encoding work?

• Steps:
  – Assign a value from 0 to 7 to each pixel based on the closest of the 8 coordinate values V0, V1, ..., V7
  • Creates a 4x4 3-bit lookup table for storage
How does 3Dc decoding work?

- **Steps:**
  - For each block in the x and y channels, replace values from lookup table with one of the 8 coordinate values (2 stored values and 6 interpolated values)
  - Compute z based on x and y to get all three coordinates for each normal vector
3Dc Compression Rate

- Suppose we are given an 4x4 normal map, with each pixel represented by x, y, and z values ranging from 0 to $2^{16}-1$ each.
- The amount of bits required to store this image in an uncompressed format is $4\times4\times(3\times16\text{bits})=768\text{ bits}$
- The bit rate of the normal map in an uncompressed format is 48 bpp (bits per pixel)
3Dc Compression Rate

- Supposed we compress the normal map using 3Dc
- The high and low representative coordinate values V0 and V1 each require 8 bits
- Each value in the 4x4 lookup table represents 8 possible values, thus requiring $4 \times 4 \times 3 \times \text{bit}=48 \text{ bits}$
3Dc Compression Rate

- 2 of the three channels must be stored (i.e., 2 lookup tables, 2 sets of V0 and V1, etc.)
- The amount of bits required to store this color image in 3Dc compressed format is 
  \((2 \times 8 \text{ bits} + 48 \text{ bits}) \times 2 = 128 \text{ bits}\)
- The bit rate of the normal map in a 3Dc compressed format is \(128/16 = 8\text{bpp}\)
- Effective compression rate for 3Dc in this case is: 
  - \(48/8 = 6:1\) compression
3Dc Compression Results