MANCHESTER 1824

COMP27112

Computer Graphics and Image Processing

9: Surface detail

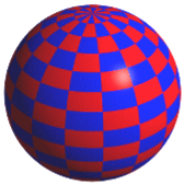
Toby.Howard@manchester.ac.uk

1

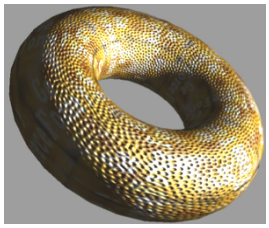
MANCHESTER 1824

Surface detail

- We'll look at two methods for adding **surface detail** to rendered surfaces
  - Texture mapping



  - Bump mapping



2

MANCHESTER 1824

Defining a texture

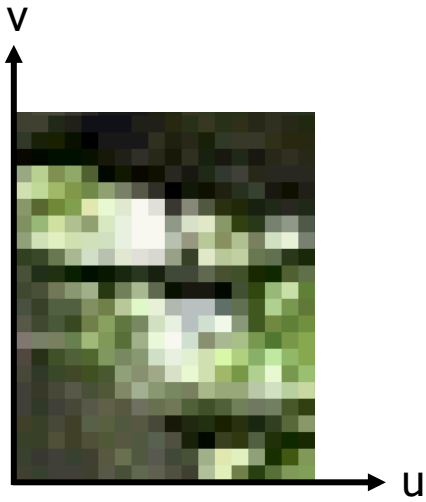
- A texture is defined as a 2D array of "texels", often read from an image file

3

MANCHESTER 1824

Texture coordinates

- A texture is defined in its own coordinate system, conventionally referred to as (u,v)

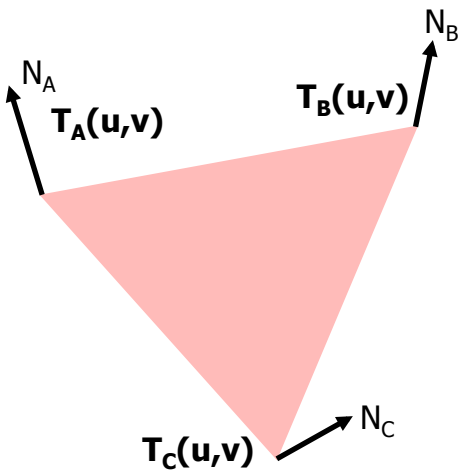


4

MANCHESTER 1824

Mapping texture per-polygon

- We associate (u,v) texture coordinates with each (x,y,z) vertex of a polygon
- And interpolate the texture coordinates during scan-conversion
- Then we **blend** the pixel colour with the texture colour

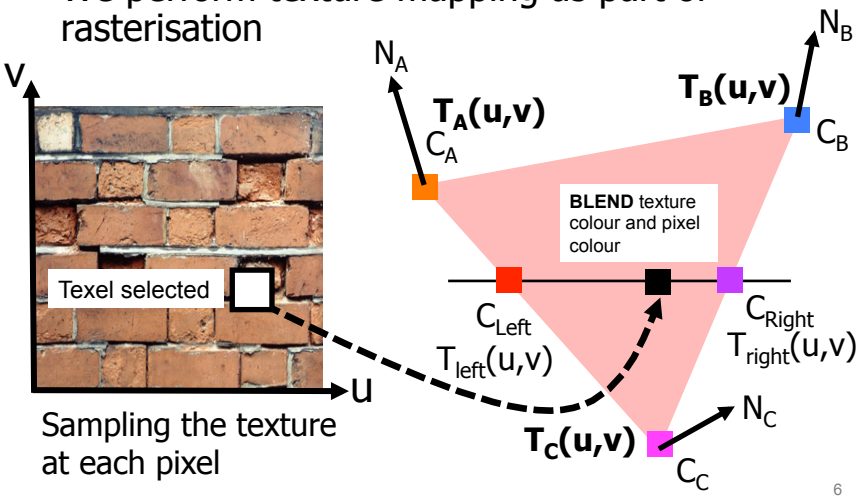


5

MANCHESTER 1824

Performing texture mapping

- We perform texture mapping as part of rasterisation



Sampling the texture at each pixel

6

MANCHESTER 1824

Meshes and seams

- In order to realistically texture a mesh, we often have to use multiple textures in different places, which can give rise to ugly "seams"

- One solution is to use textures that are "seamless", so the edge of one exactly matches the edge of another

MANCHESTER 1824

Resolution mismatches

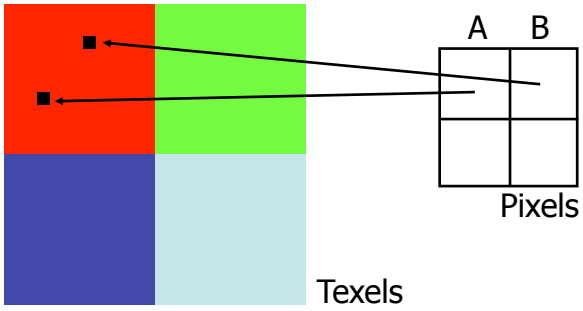
- We have to deal with two situations:
  - When pixel resolution **>** texture resolution (i.e. texels are bigger than pixels). So we need to **filter** the texture
  - When texture resolution **>** pixel resolution (i.e. pixels are bigger than texels). So we need to **filter** the texture
- Each situation requires a different approach

8

MANCHESTER 1824

pixel res. > texel res.

- Pixels A and B happen to map to the same texel, because pixel resolution > texel resolution



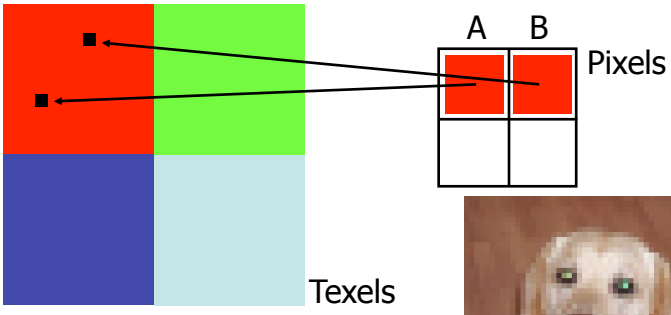
- We'll look at two approaches (there are more):
  1. No filter
  2. Bilinear interpolation filter

9

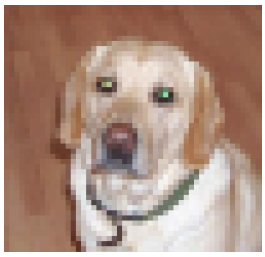
MANCHESTER 1824

1. No filter

- We simply select the texel to which the pixel maps



- The resulting pixel image looks blocky



10

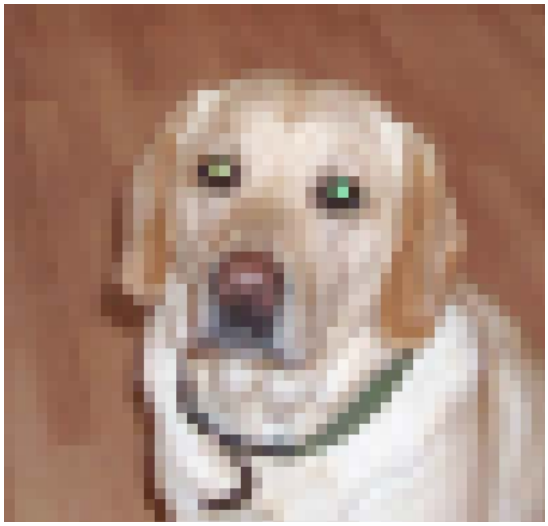
MANCHESTER 1824

Pixel resolution == Texel resolution

11

MANCHESTER 1824

Pixel res. > Texel res. : no filter

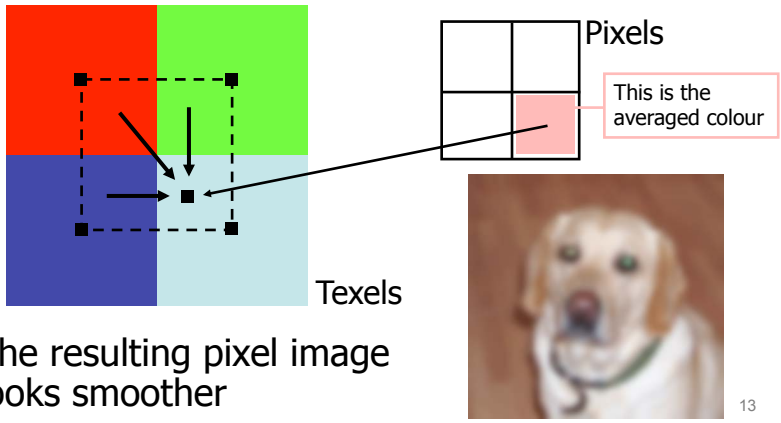


12

MANCHESTER 1824

2. Bilinear interpolation filter

- We compute a texel colour from adjacent texels, **averaging** horizontally and vertically



- The resulting pixel image looks smoother

13

MANCHESTER 1824

Pixel resolution == Texel resolution

14

MANCHESTER 1824

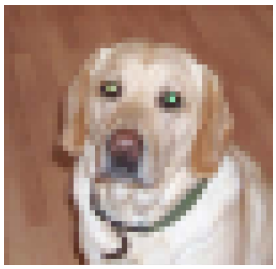
Pixel res. > Texel res. : bilinear filter

15

MANCHESTER 1824

Texture filtering: comparison

Unfiltered texture (1 to 1 pixel correspondence)



No filter

Bilinear interpolation filter

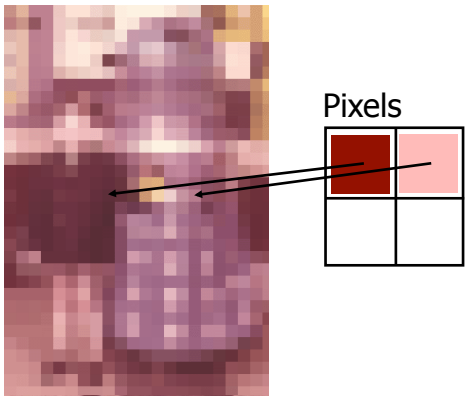
16

MANCHESTER 1824

texel resolution > pixel resolution

- If texel resolution **>** pixel resolution, **adjacent** pixels may map to texels **far apart** in the texture, leading to "missing detail" (aliasing)

- In animated sequences especially, we see unpleasant aliasing effects, as pixels "pop" on and off, or change colour unexpectedly in each frame



17

MANCHESTER 1824

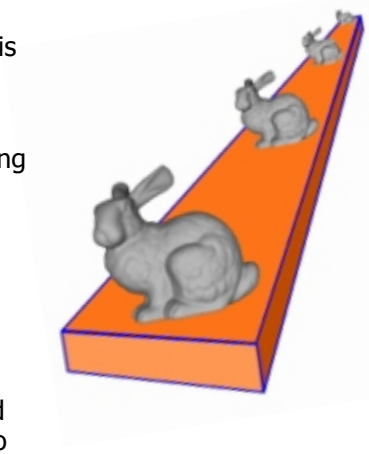
Mipmap filtering

- One technique to minimise this effect is "mipmapping"

- mip = *multum in parvo* (Latin), meaning "many things in a small place"

- The idea is simple: the further away from the viewpoint, the less detail we need

- So, we use a **set** of texture maps, and **select** which map to use, according to the **distance** of a pixel from the viewer



18

MANCHESTER 1824

Creating a mipmap

t0 t1 t2 t3 t4 (etc)

t0 is the original full-resolution texture

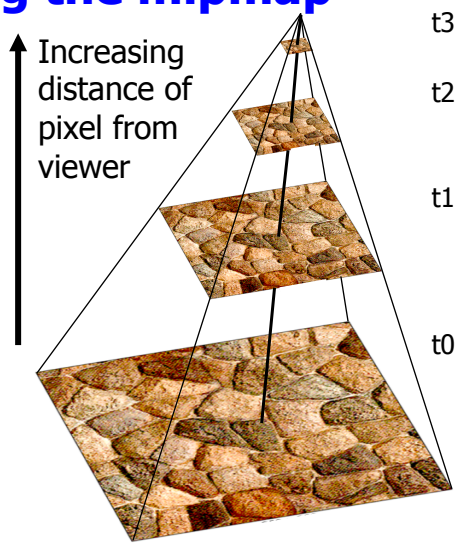
Starting from the original texture (t0), we repeatedly create smaller versions of it (t1, t2 etc), downsampling by ½ each time, until we reach a 1x1 texture. This is a **pre-processing** procedure, and we store each texture in memory.

19

MANCHESTER 1824

Rendering using the mipmap

- When rendering, we select one of the textures according to the **distance** of the pixel from the viewpoint
- Alternatively, we can also choose the **two closest** textures, and do bilinear interpolation – for extra smoothness

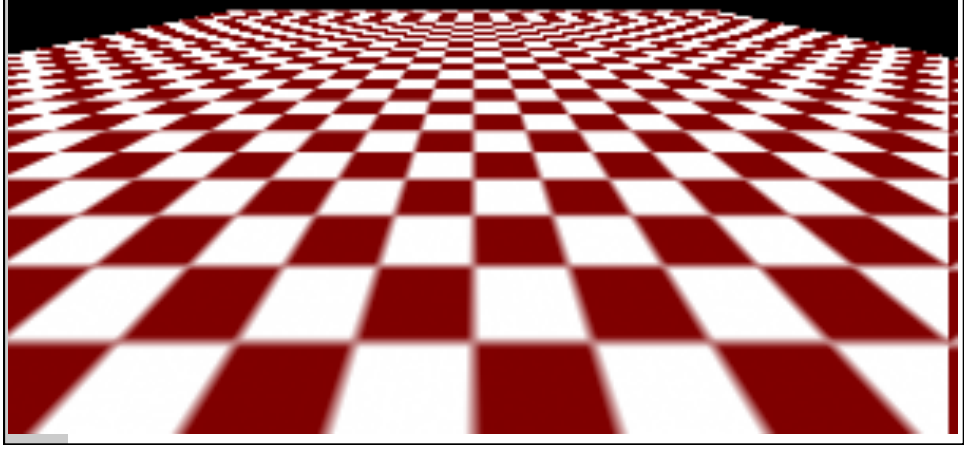


20

MANCHESTER 1824

Mipmaps

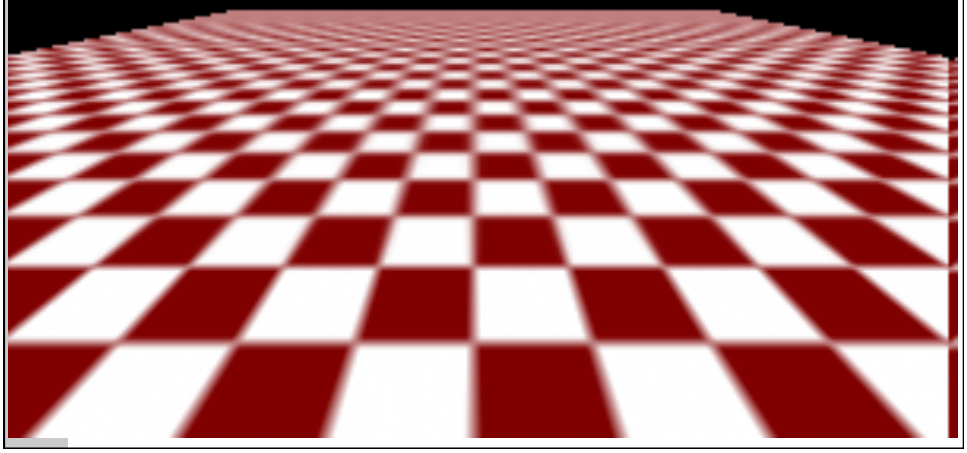
With no mipmapping, we see aliasing on distant polygons



MANCHESTER 1824

Mipmaps

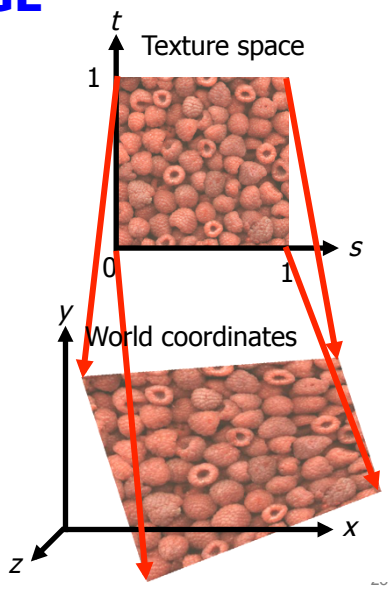
With mipmapping, the aliasing is replaced by blurring



MANCHESTER 1824

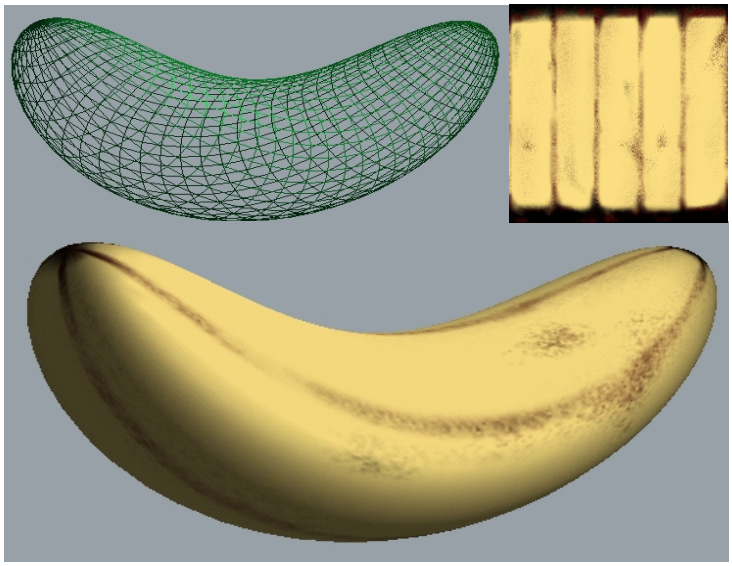
Textures in OpenGL

```
/* First call functions to
   read texture from
   image file, and to set
   its properties - how
   it affects pixels */

/* Now use it */

glBegin(GL_QUADS);
 glTexCoord(0.0, 0.0);
   glVertex3f(x0, y0, z0);
 glTexCoord(1.0, 0.0);
   glVertex3f(x1, y1, z1);
 glTexCoord(1.0, 1.0);
   glVertex3f(x2, y2, z2);
 glTexCoord(0.0, 1.0);
   glVertex3f(x3, y3, z3);
glEnd();
```



MANCHESTER 1824

Texture example



James Sinnott

24

MANCHESTER 1824

Textures as "illumination"

- Textures can be used to add accurate illumination to a real-time scene:
  - Off-line, compute accurate diffuse illumination of the scene, using a global model (radiosity)
  - Save rendered surfaces as textures – called **lightmaps**
- In real-time, apply lightmap textures, and also actual surface textures.

25

MANCHESTER 1824

Lightmaps only

26

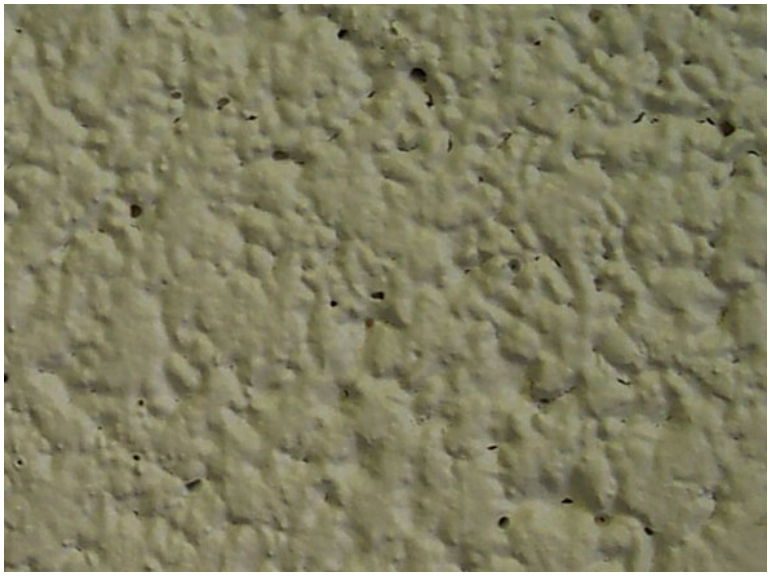
MANCHESTER 1824

Lightmaps blended with other textures

27

MANCHESTER 1824

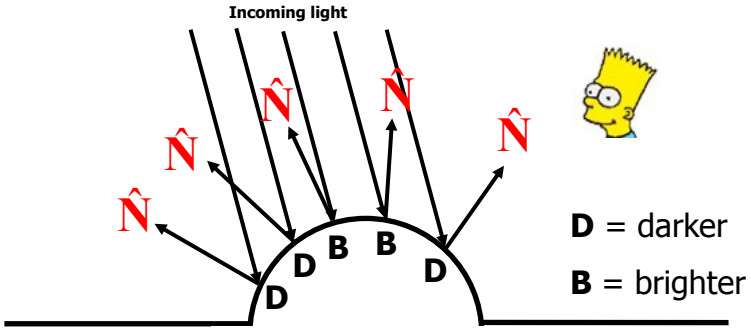
Modelling bumpy surfaces



28

MANCHESTER 1824

Bump mapping



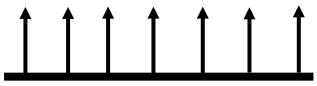
- Why bumpy surfaces look bumpy
  - The **surface normals** change across the bumps, so the top of the bumps appear brighter than the sides

29

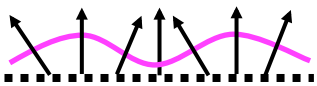
MANCHESTER 1824

Bump mapping

- Rather than alter the surface **colour**, as in texturing, we can **alter** the **surface normal**…
- …which has the same effect on the illumination model as if the surface were really geometrically different



Original normals

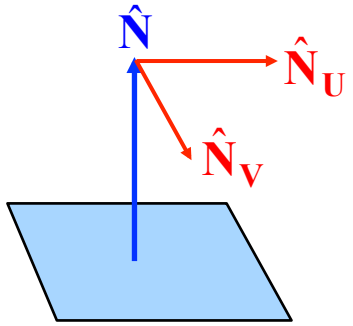


Altered normals

30

MANCHESTER 1824

Altering a surface normal

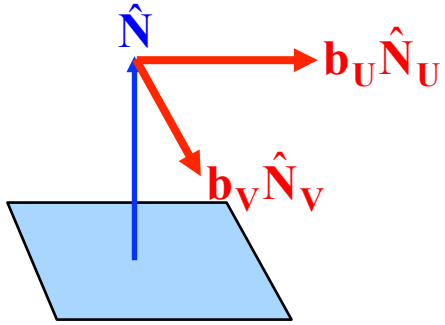


- Step 1: Introduce two **unit** vectors $\hat{N}_U$ and $\hat{N}_V$ each orthogonal to the surface normal

31

MANCHESTER 1824

Altering a surface normal

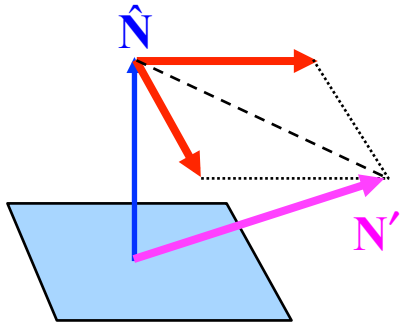


- Step 2: Scale $\hat{N}_U$ and $\hat{N}_V$ by b_U and b_V
- To give the "bump vectors"

32

MANCHESTER 1824

Altering a surface normal

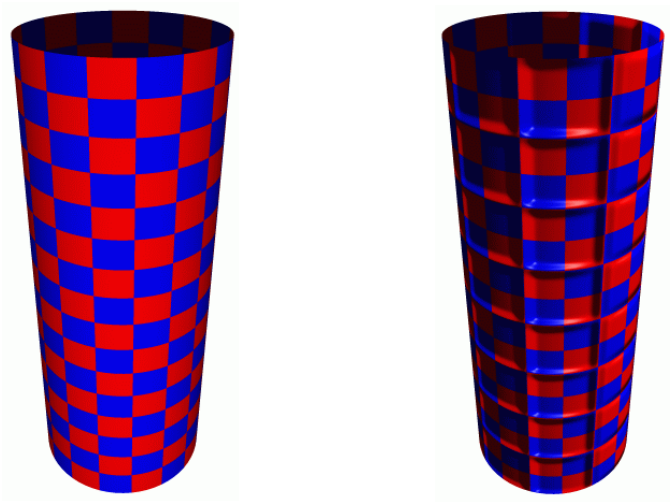


- Step 3: Add the bump vectors to the original surface normal to get an altered surface normal
- $N' = \hat{N} + (b_U \hat{N}_U + b_V \hat{N}_V)$

33

MANCHESTER 1824

Bump mapping example



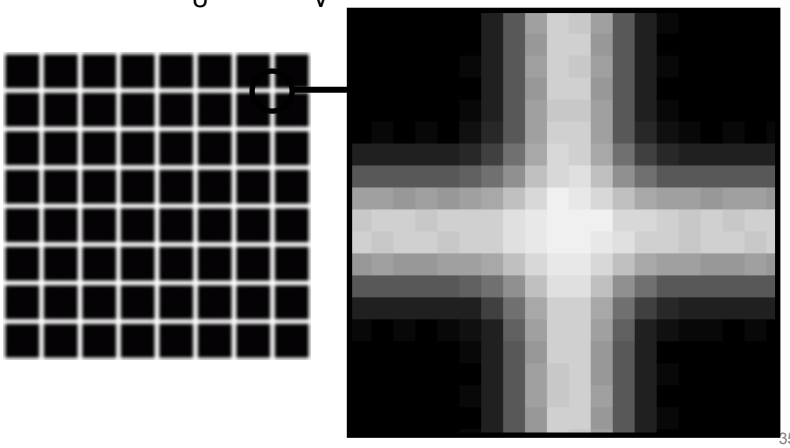
Leonard McMillan

34

MANCHESTER 1824

Deriving b_U and b_V

- We can draw a texture to use as a **bump map** to derive b_U and b_V



Leonard McMillan

35

MANCHESTER 1824

Format of a bump map

- We treat the value of each texel as a "height" which controls the bumpiness of the surface
- Here, the brighter texels are "higher" than the darker texels

4	6	8	10	8	6	4	4	4
4	6	8	10	8	6	4	4	4
4	6	8	10	8	6	4	4	4
4	6	8	10	8	6	6	6	6
4	6	8	10	8	8	8	8	8
4	6	8	10	10	10	10	10	10
4	4	6	8	8	8	8	8	8
4	4	4	6	6	6	6	6	6
4	4	4	4	4	4	4	4	4

Hugo Elias

36

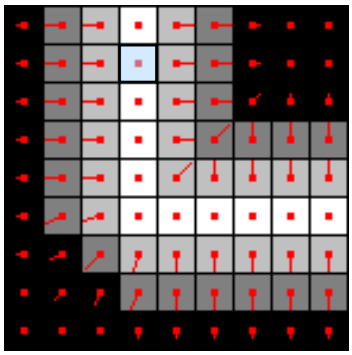
MANCHESTER 1824

Deriving b_U and b_V

- For each texel T, we can compute its x and y **gradients** and use them as b_U and b_V:
  - $b_U = T(x-1, y) - T(x+1, y)$
  - $b_V = T(x, y-1) - T(x, y+1)$

(0,0)

4	6	8	10	8	6	4	4	4
4	6	8	10	8	6	4	4	4
4	6	8	10	8	6	4	4	4
4	6	8	10	8	6	6	6	6
4	6	8	10	8	8	8	8	8
4	6	8	10	10	10	10	10	10
4	4	6	8	8	8	8	8	8
4	4	4	6	6	6	6	6	6
4	4	4	4	4	4	4	4	4



Hugo Elias

37

MANCHESTER 1824

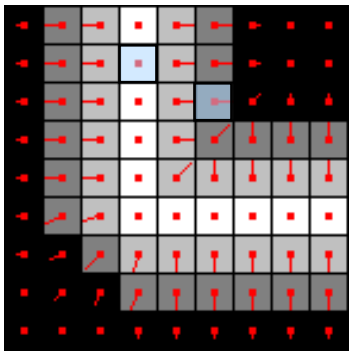
Deriving b_U and b_V

- For each texel T, we can compute its x and y **gradients** and use them as b_U and b_V:
  - $b_U = T(x-1, y) - T(x+1, y)$
  - $b_V = T(x, y-1) - T(x, y+1)$

(0,0)

(4,0)

4	6	8	10	8	6	4	4	4
4	6	8	10	8	6	4	4	4
4	6	8	10	8	6	4	4	4
4	6	8	10	8	6	6	6	6
4	6	8	10	8	8	8	8	8
4	6	8	10	10	10	10	10	10
4	4	6	8	8	8	8	8	8
4	4	4	6	6	6	6	6	6
4	4	4	4	4	4	4	4	4



Hugo Elias

38

MANCHESTER 1824

Deriving b_U and b_V

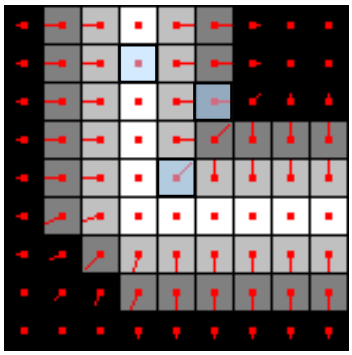
- For each texel T, we can compute its x and y **gradients** and use them as b_U and b_V:
  - $b_U = T(x-1, y) - T(x+1, y)$
  - $b_V = T(x, y-1) - T(x, y+1)$

(0,0)

(4,0)

(2,2)

4	6	8	10	8	6	4	4	4
4	6	8	10	8	6	4	4	4
4	6	8	10	8	6	4	4	4
4	6	8	10	8	6	6	6	6
4	6	8	10	8	8	8	8	8
4	6	8	10	10	10	10	10	10
4	4	6	8	8	8	8	8	8
4	4	4	6	6	6	6	6	6
4	4	4	4	4	4	4	4	4



Hugo Elias

39

MANCHESTER 1824

Deriving b_U and b_V

- For each texel T, we can compute its x and y **gradients** and use them as b_U and b_V:
  - $b_U = T(x-1, y) - T(x+1, y)$
  - $b_V = T(x, y-1) - T(x, y+1)$

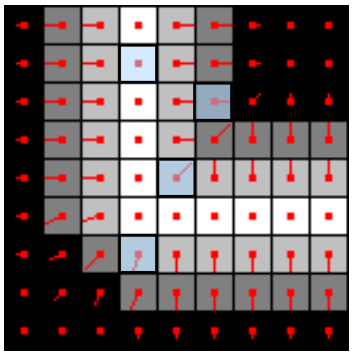
(0,0)

(4,0)

(2,2)

(-2,-4)

4	6	8	10	8	6	4	4	4
4	6	8	10	8	6	4	4	4
4	6	8	10	8	6	4	4	4
4	6	8	10	8	6	6	6	6
4	6	8	10	8	8	8	8	8
4	6	8	10	10	10	10	10	10
4	4	6	8	8	8	8	8	8
4	4	4	6	6	6	6	6	6
4	4	4	4	4	4	4	4	4



Hugo Elias

40

MANCHESTER 1824

Bump mapping example

NVIDIA

41