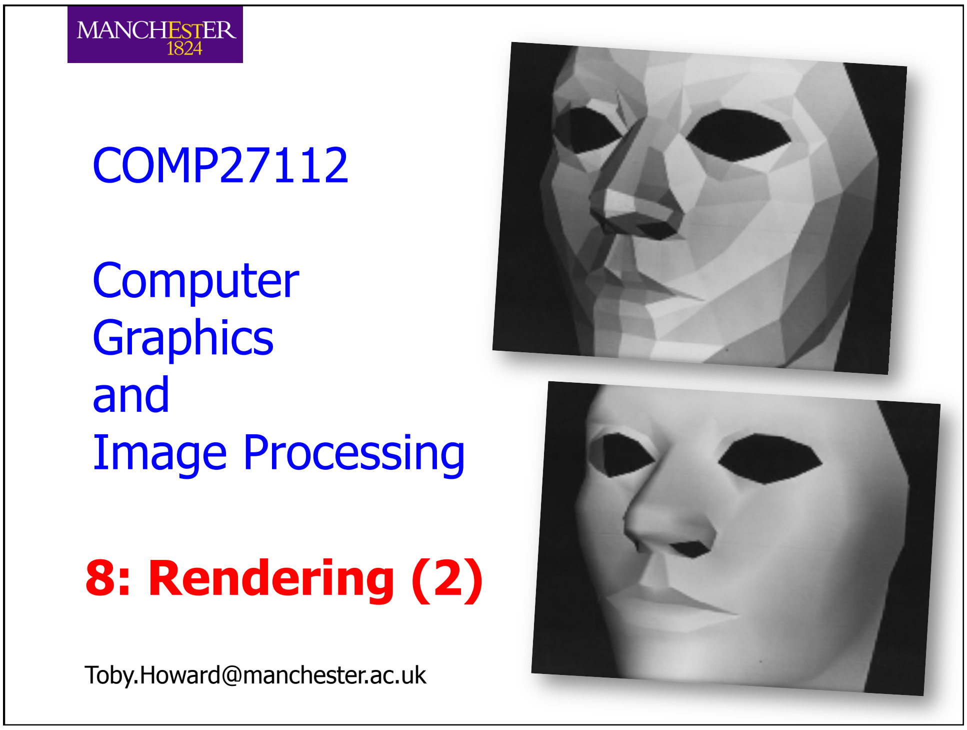


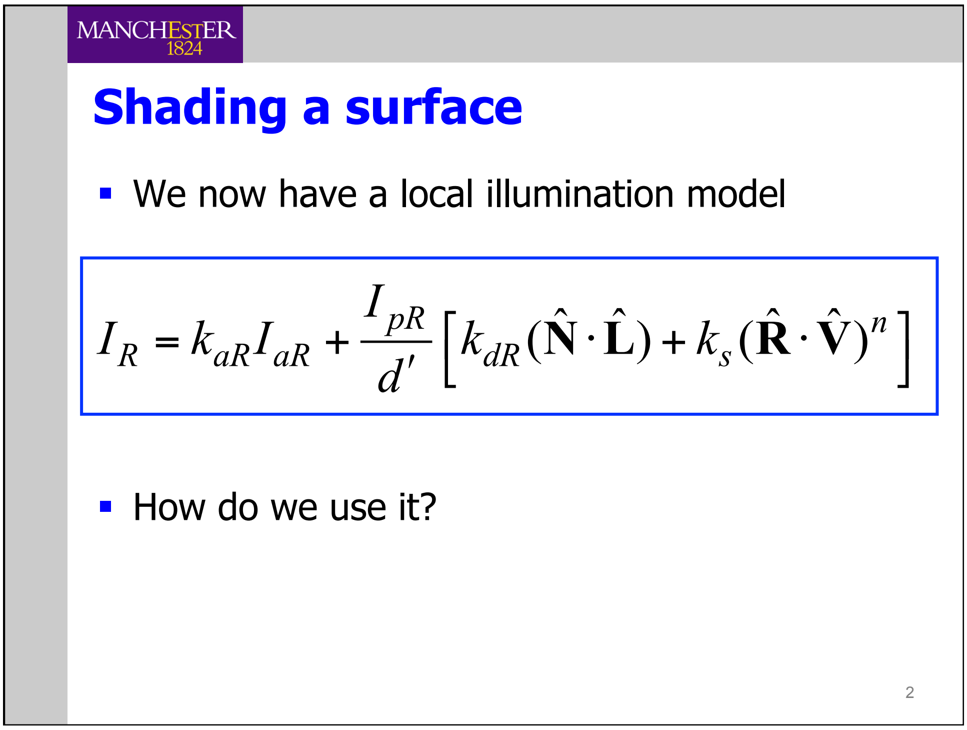
MANCHESTER 1824

COMP27112

Computer Graphics and Image Processing

8: Rendering (2)

Toby.Howard@manchester.ac.uk



MANCHESTER 1824

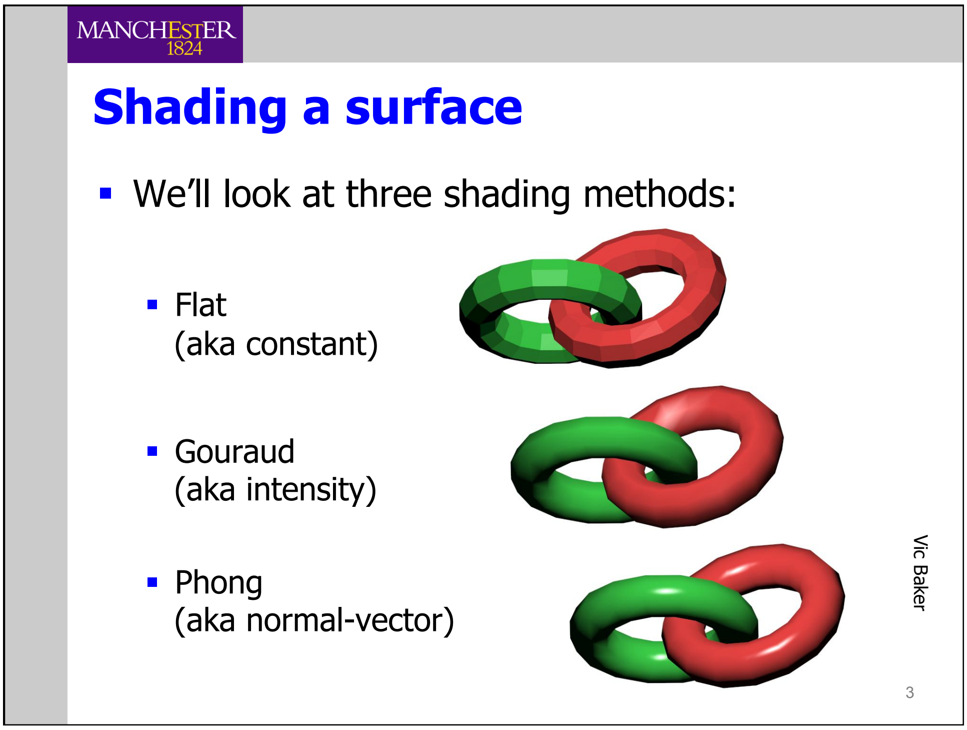
Shading a surface

- We now have a local illumination model

$$I_R = k_{aR} I_{aR} + \frac{I_{pR}}{d'} \left[k_{dR} (\hat{\mathbf{N}} \cdot \hat{\mathbf{L}}) + k_s (\hat{\mathbf{R}} \cdot \hat{\mathbf{V}})^n \right]$$

- How do we use it?

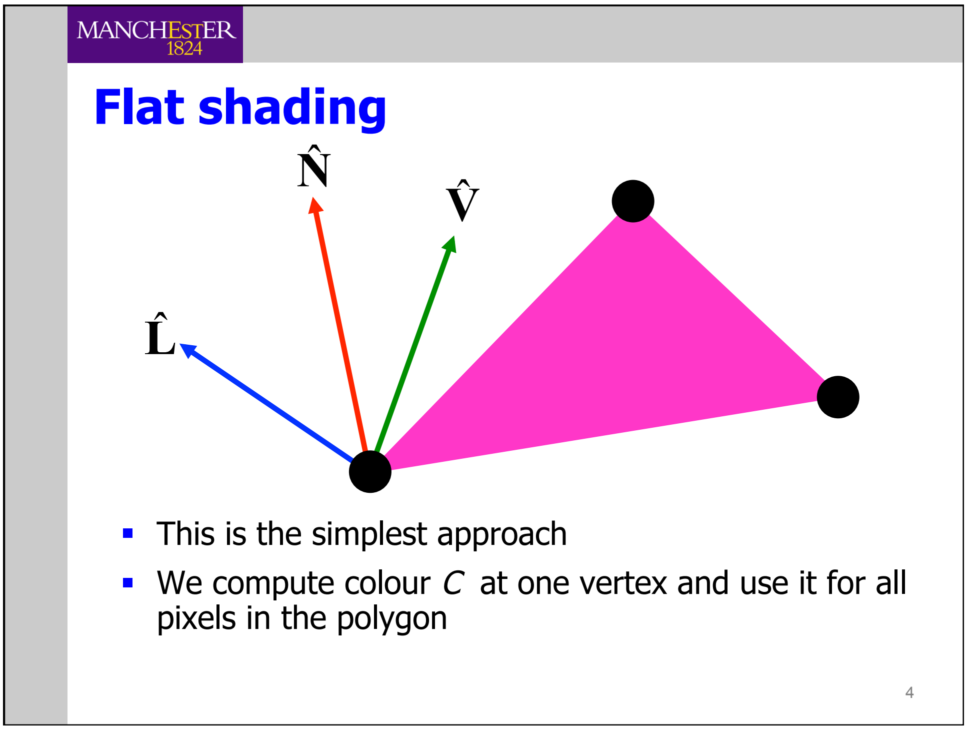
2



Shading a surface

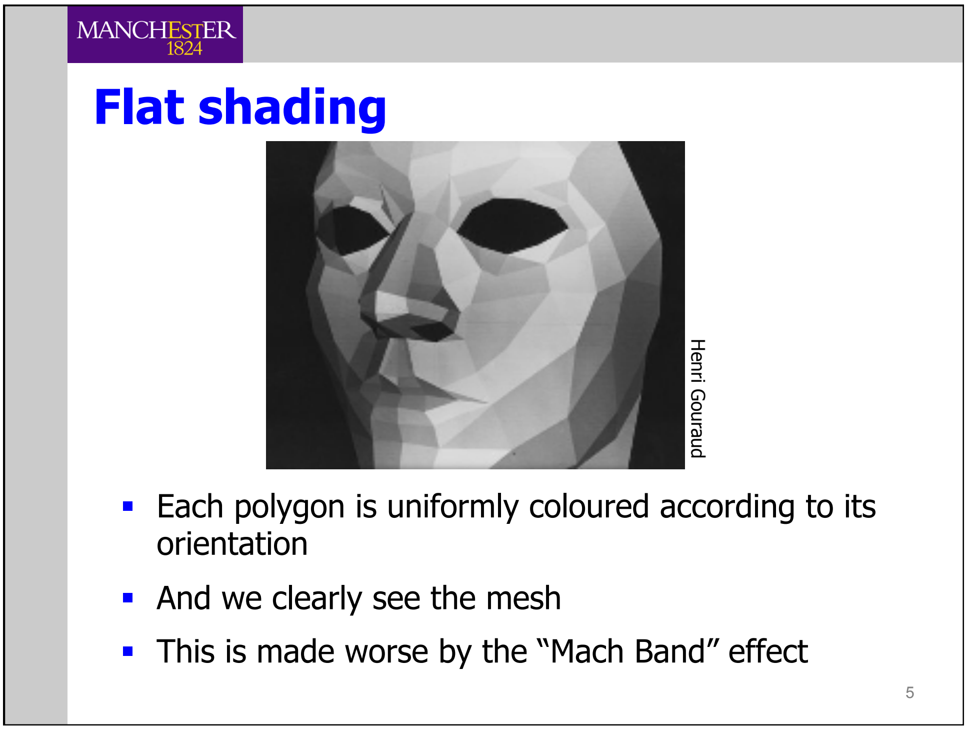
- We'll look at three shading methods:
  - Flat
    (aka constant)
  - Gouraud
    (aka intensity)
  - Phong
    (aka normal-vector)

Vic Baker



Flat shading

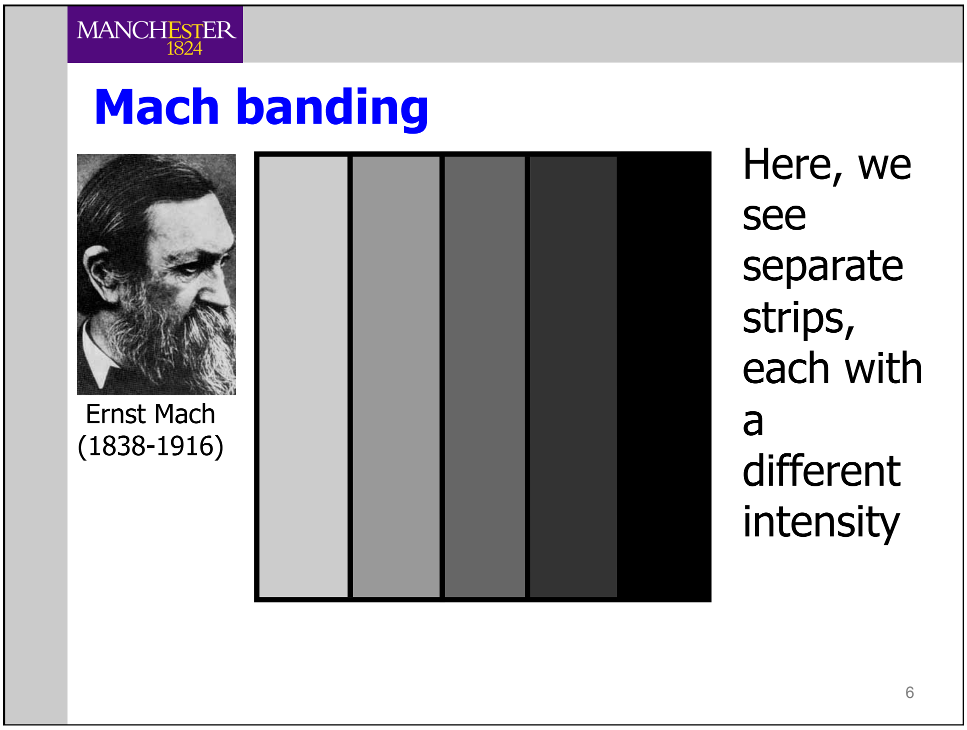
- This is the simplest approach
- We compute colour *C* at one vertex and use it for all pixels in the polygon



Flat shading

Henri Gouraud

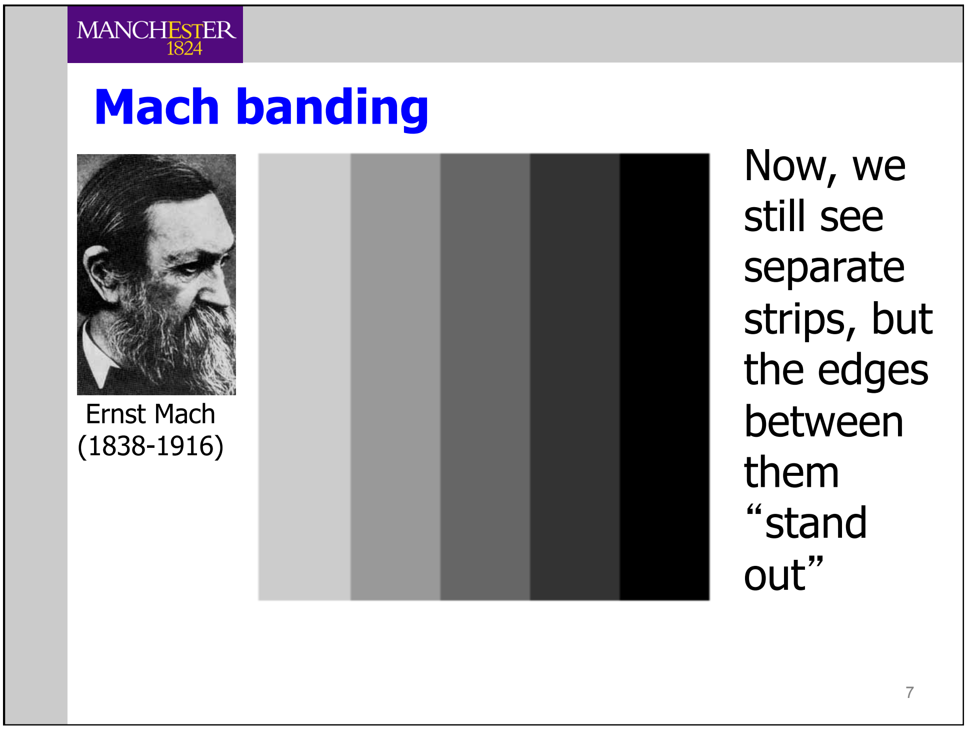
- Each polygon is uniformly coloured according to its orientation
- And we clearly see the mesh
- This is made worse by the "Mach Band" effect



Mach banding

Ernst Mach (1838-1916)

Here, we see separate strips, each with a different intensity

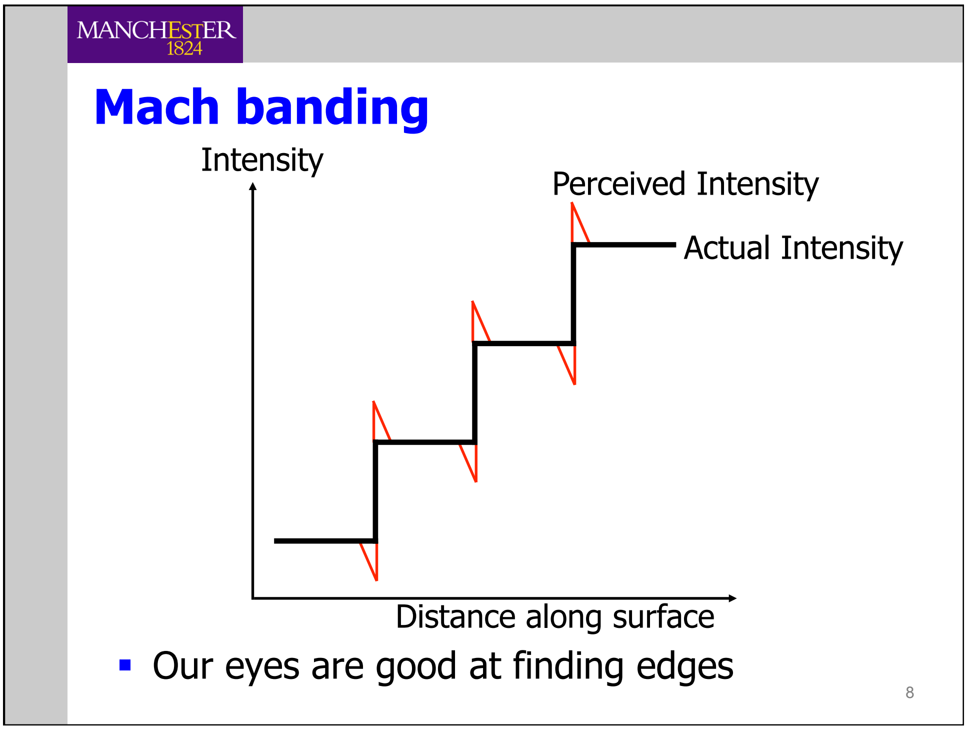


Mach banding

Ernst Mach
(1838-1916)

Now, we still see separate strips, but the edges between them “stand out”

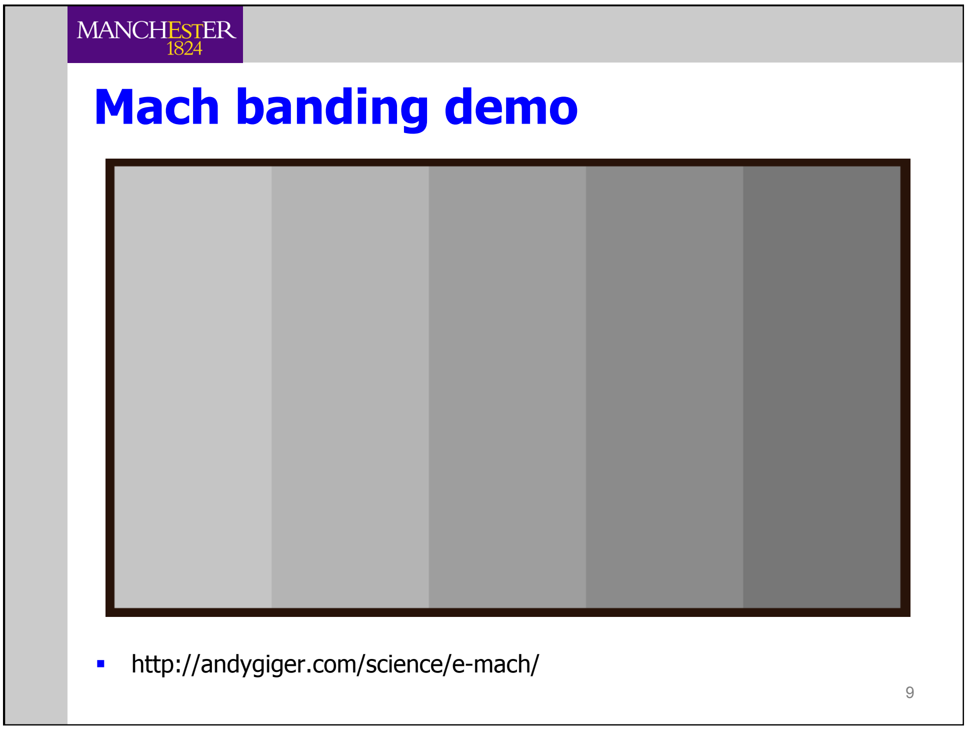
7



Mach banding

- Our eyes are good at finding edges

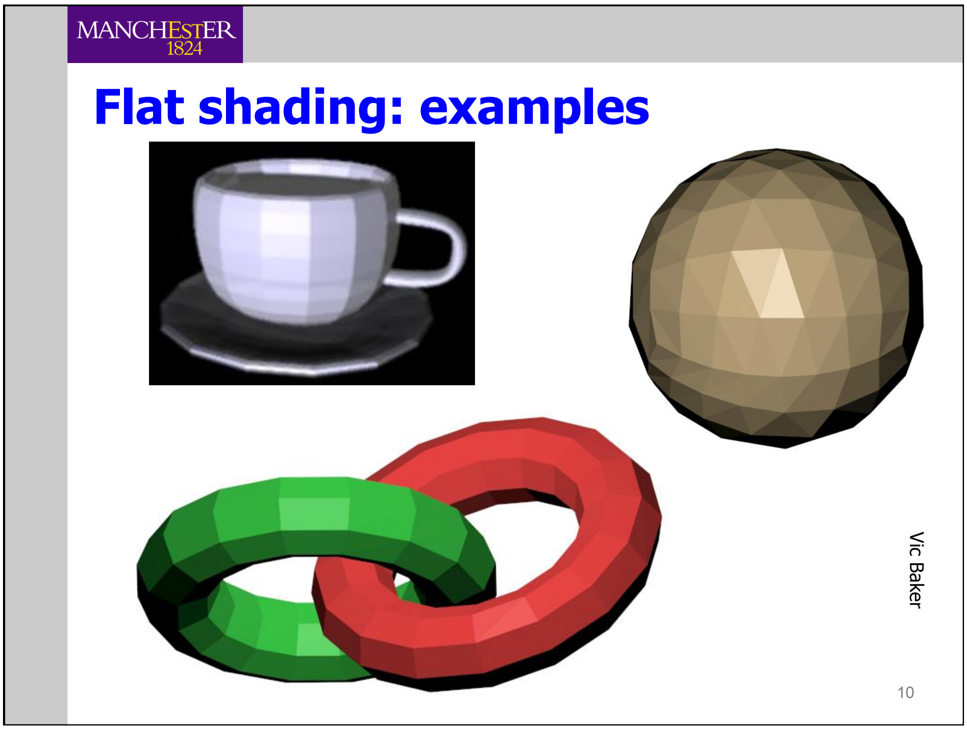
8



Mach banding demo

- http://andygiger.com/science/e-mach/

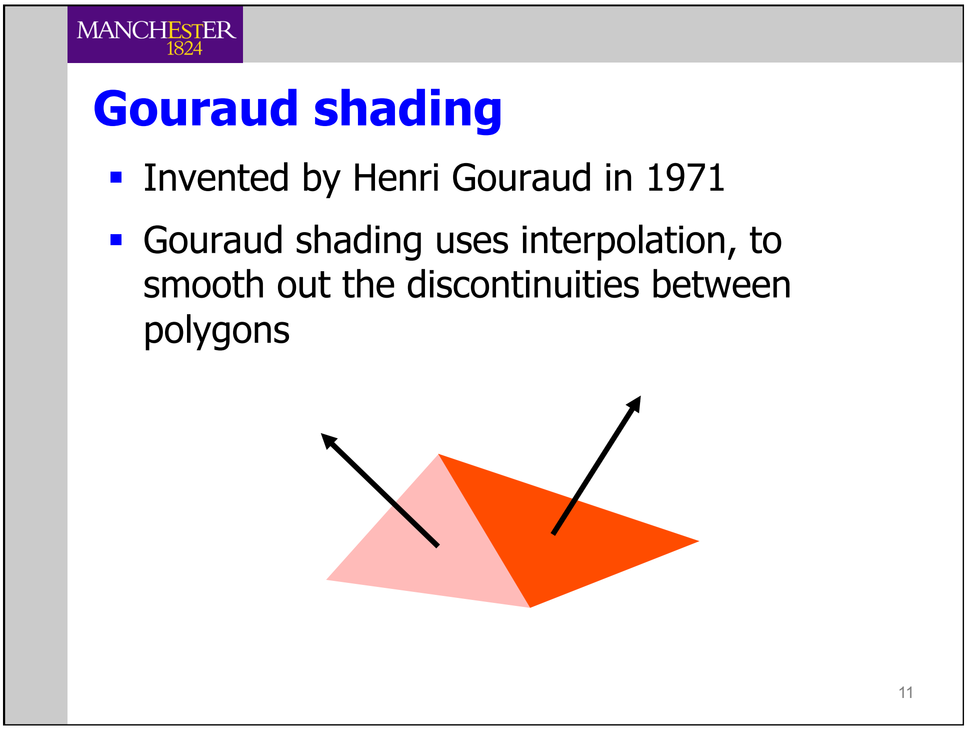
9



Flat shading: examples

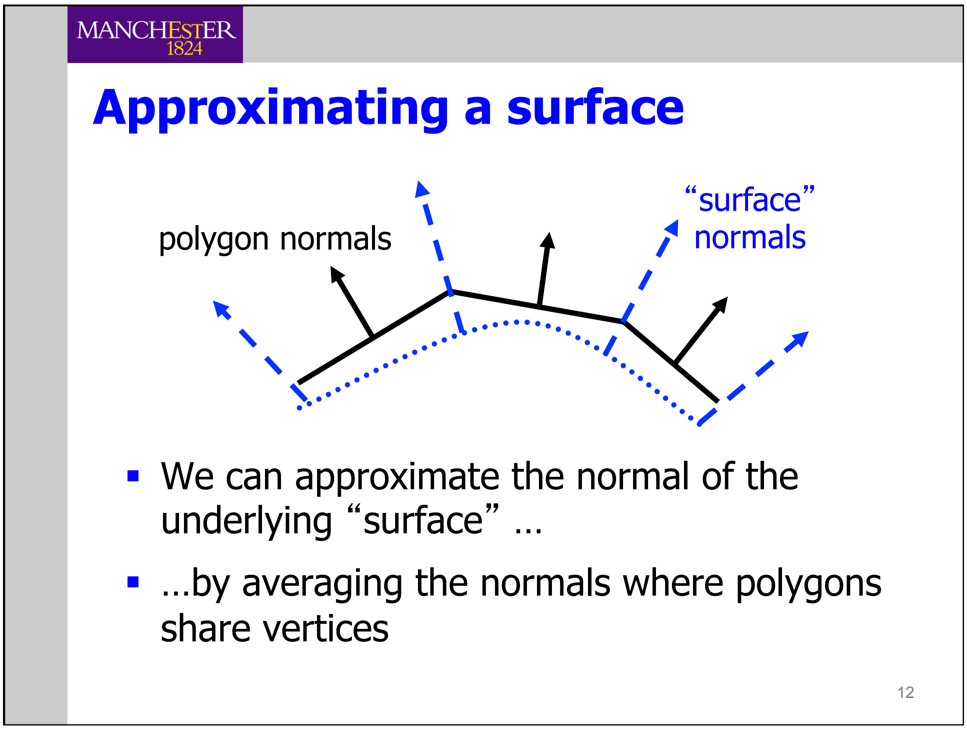
Vic Baker

10



Gouraud shading

- Invented by Henri Gouraud in 1971
- Gouraud shading uses interpolation, to smooth out the discontinuities between polygons

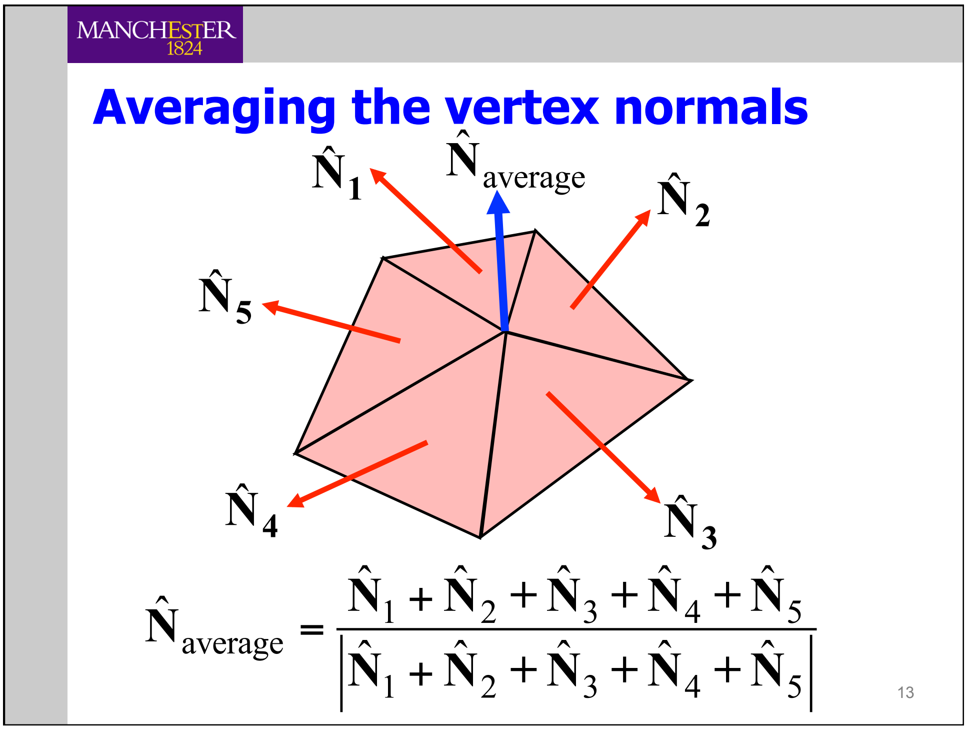


Approximating a surface

polygon normals

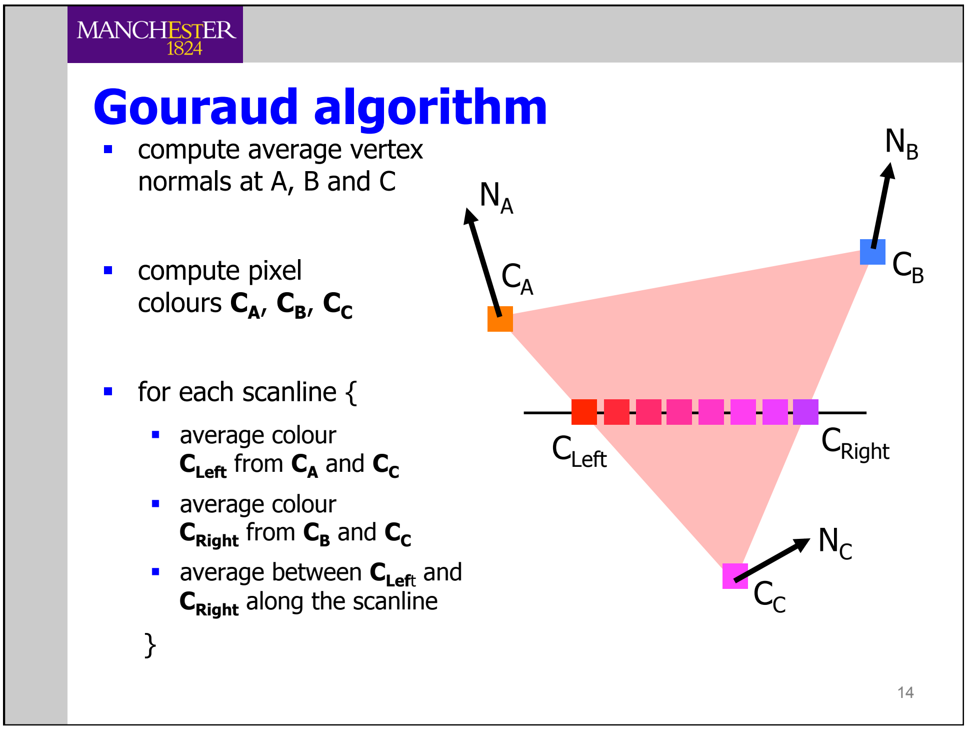
“surface” normals

- We can approximate the normal of the underlying “surface” …
- …by averaging the normals where polygons share vertices



Averaging the vertex normals

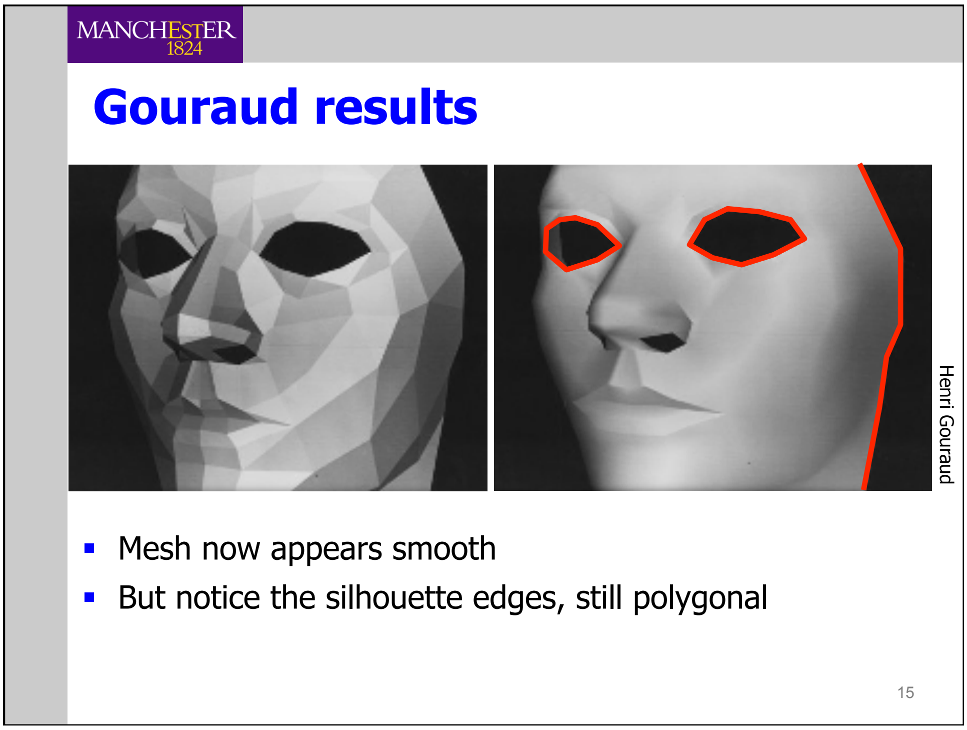
$$\hat{N}_{average} = \frac{\hat{N}_1 + \hat{N}_2 + \hat{N}_3 + \hat{N}_4 + \hat{N}_5}{\left|\hat{N}_1 + \hat{N}_2 + \hat{N}_3 + \hat{N}_4 + \hat{N}_5\right|}$$



Gouraud algorithm

- compute average vertex normals at A, B and C
- compute pixel colours $\mathbf{C_A}$, $\mathbf{C_B}$, $\mathbf{C_C}$
- for each scanline {
    - average colour $\mathbf{C_{Left}}$ from $\mathbf{C_A}$ and $\mathbf{C_C}$
    - average colour $\mathbf{C_{Right}}$ from $\mathbf{C_B}$ and $\mathbf{C_C}$
    - average between $\mathbf{C_{Left}}$ and $\mathbf{C_{Right}}$ along the scanline

  }

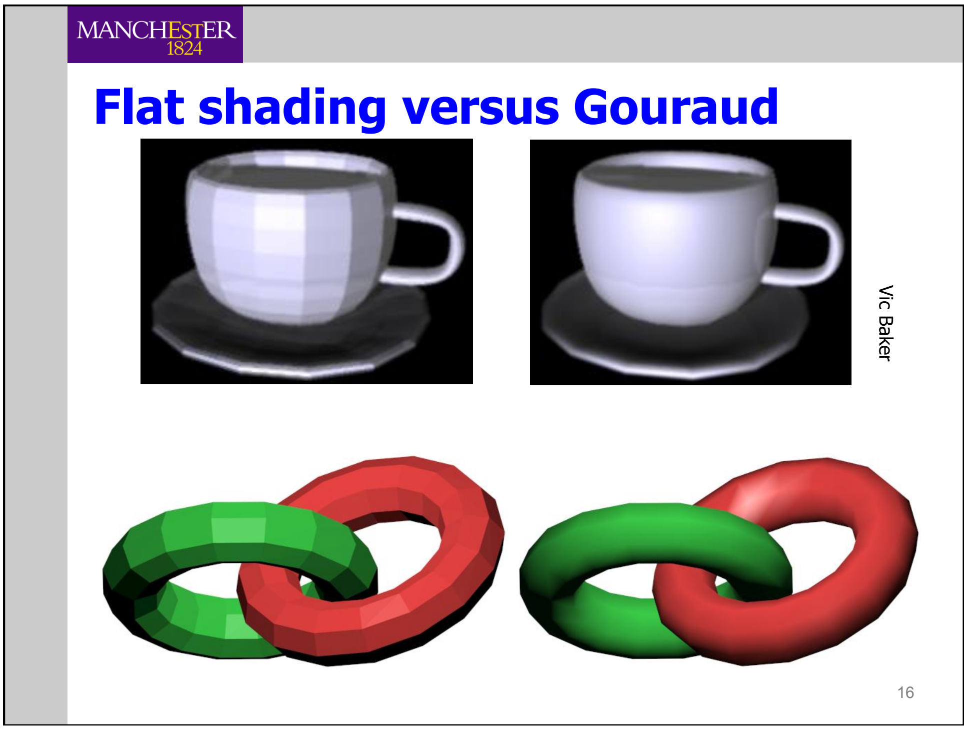


Gouraud results

Henri Gouraud

- Mesh now appears smooth
- But notice the silhouette edges, still polygonal

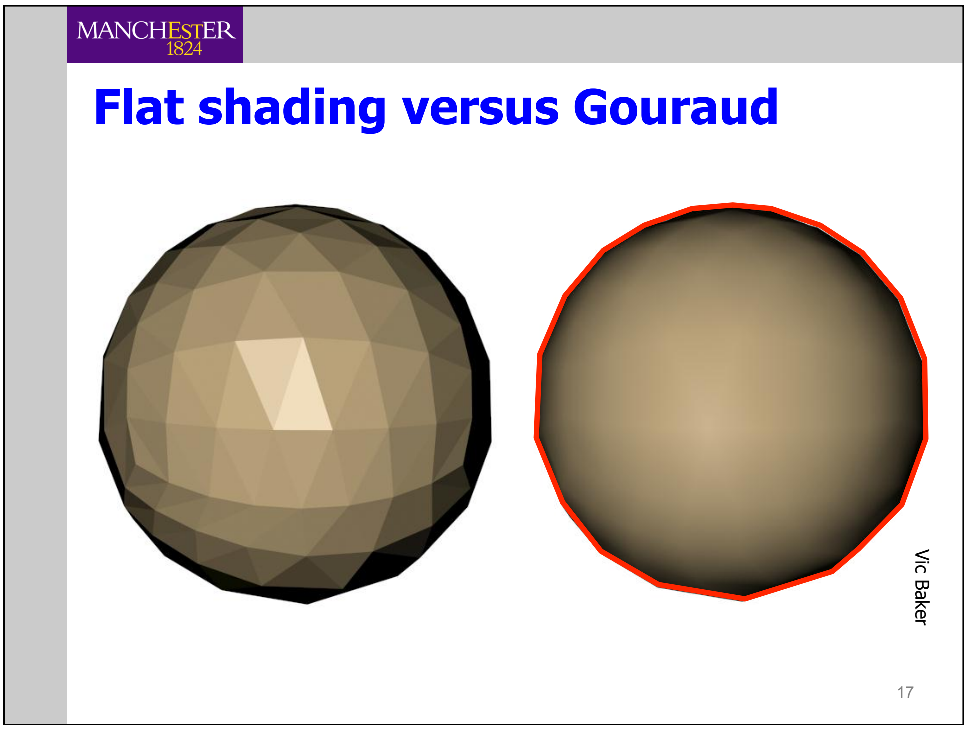
15



Flat shading versus Gouraud

Vic Baker

16

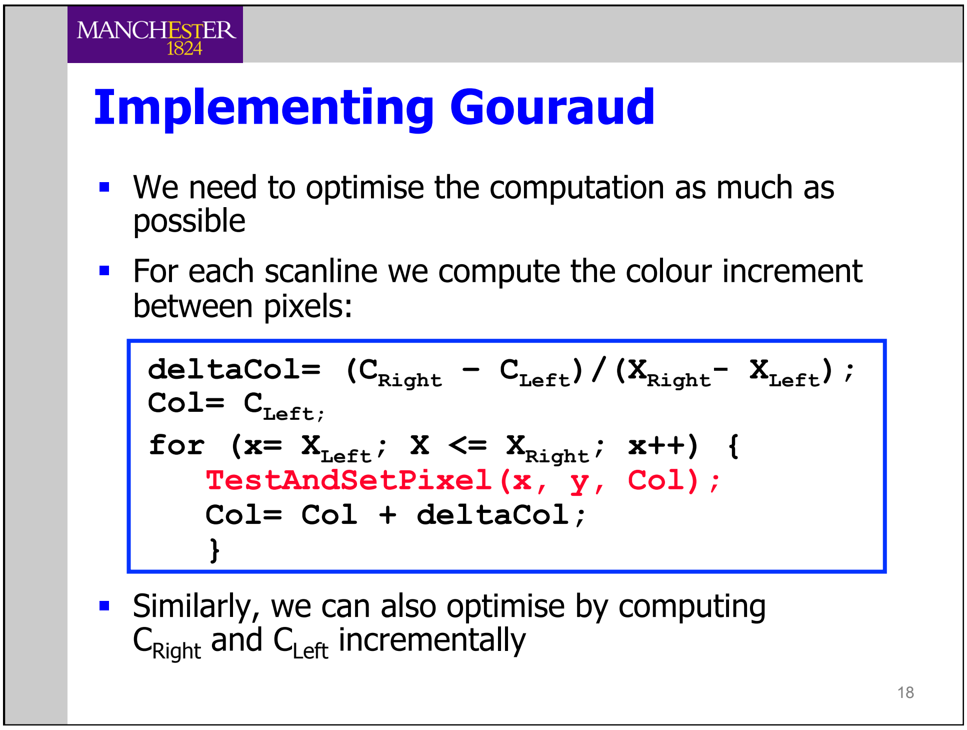


MANCHESTER 1824

Flat shading versus Gouraud

Vic Baker

17



MANCHESTER 1824

Implementing Gouraud

- We need to optimise the computation as much as possible
- For each scanline we compute the colour increment between pixels:

```
deltaCol= (C_Right - C_Left)/(X_Right- X_Left);
Col= C_Left;
for (x= X_Left; X <= X_Right; x++) {
    TestAndSetPixel(x, y, Col);
    Col= Col + deltaCol;
    }
```

- Similarly, we can also optimise by computing C_{Right} and C_{Left} incrementally

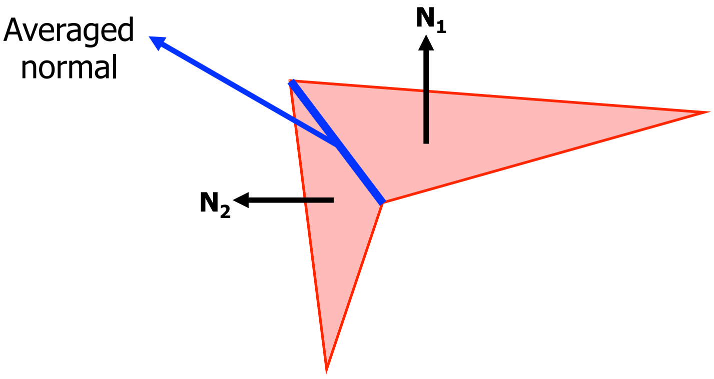
18

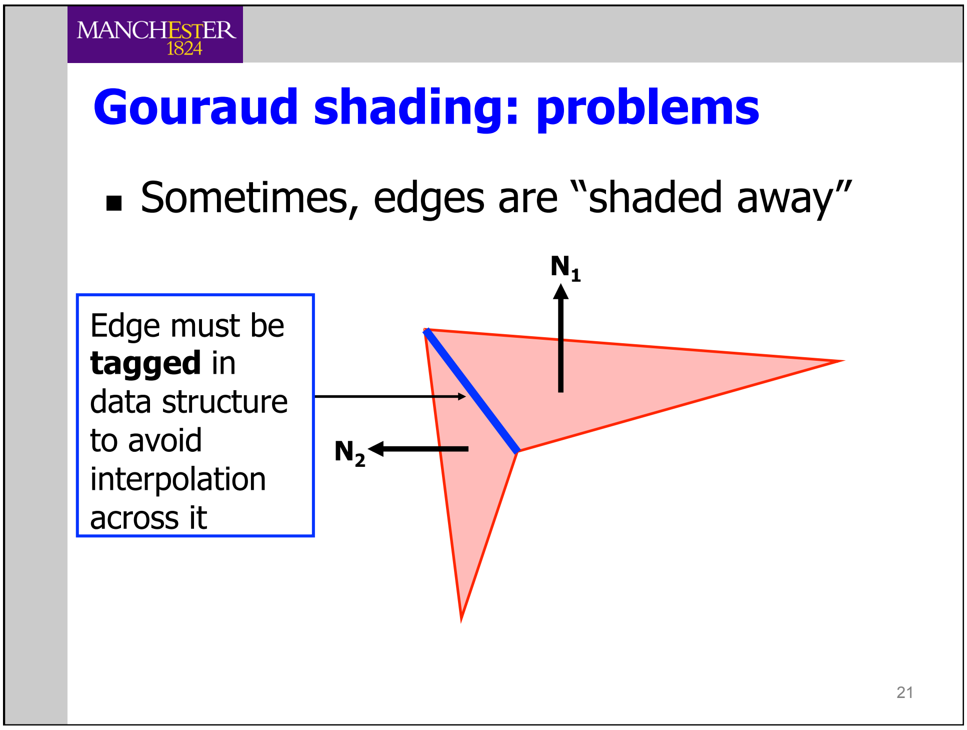
Gouraud shading: problems

- While it's fast and efficient, the method has drawbacks:
  - Specular highlights may be distorted or averaged away altogether (because Gouraud shading averages between **vertex** colours)
  - Mach banding may still be visible

Gouraud shading: problems

- Sometimes, edges are "shaded away"



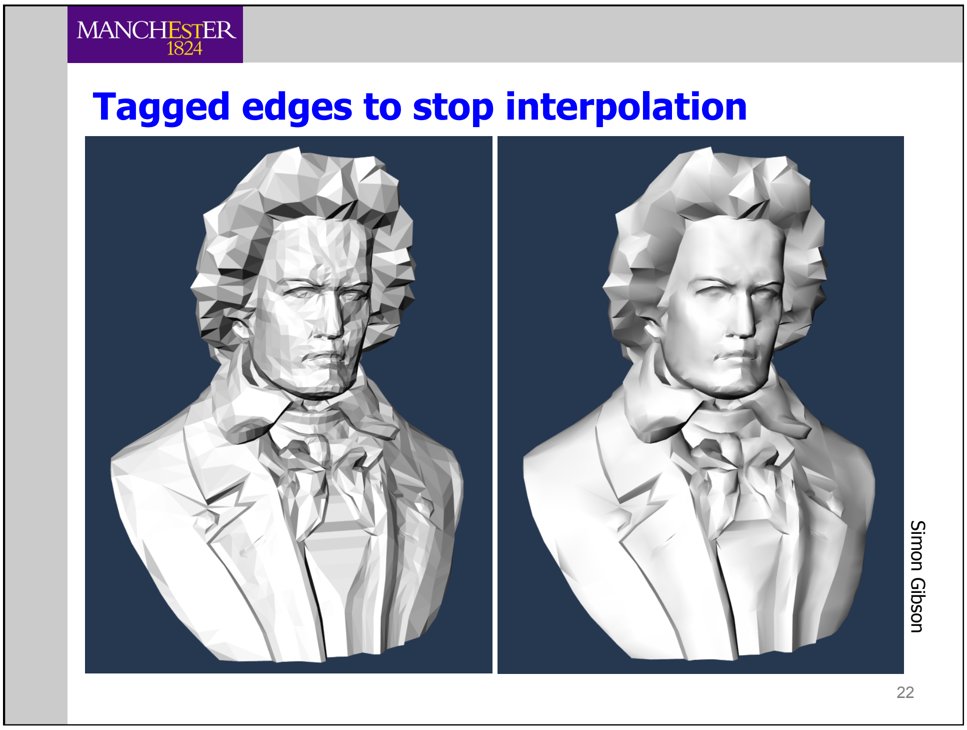


MANCHESTER 1824

Gouraud shading: problems

- Sometimes, edges are "shaded away"

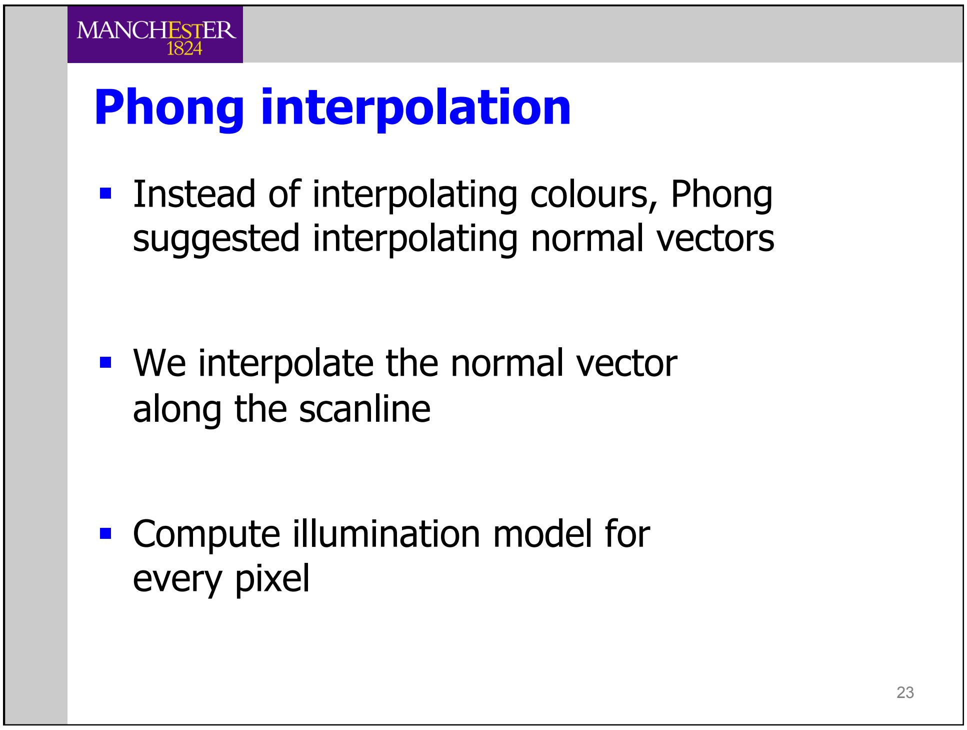
Edge must be **tagged** in data structure to avoid interpolation across it



MANCHESTER 1824

Tagged edges to stop interpolation

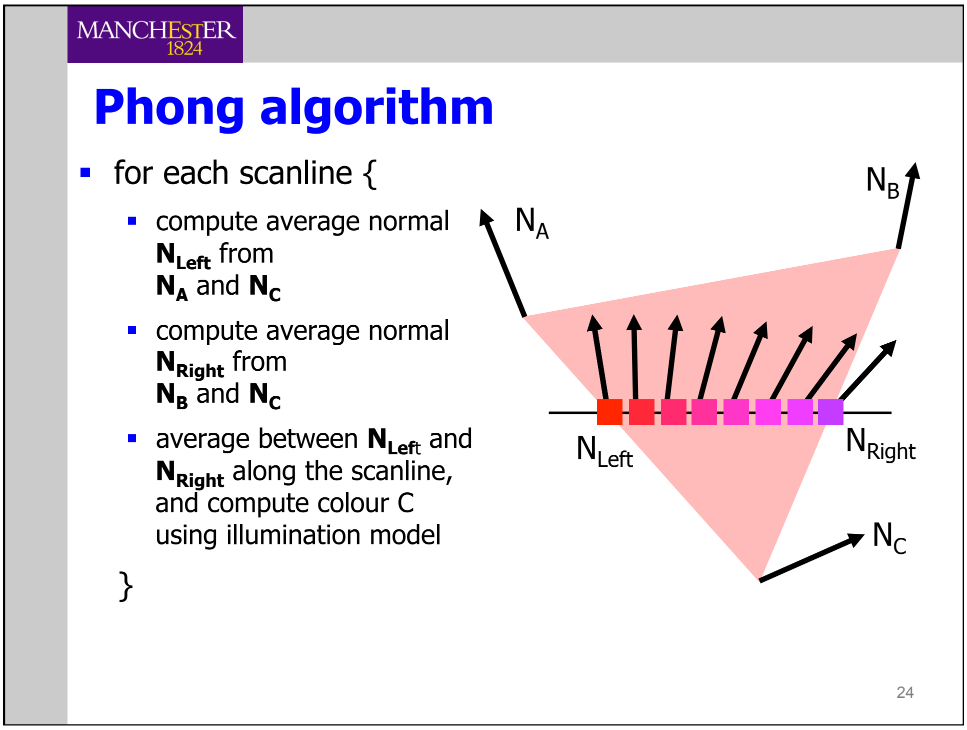
Simon Gibson



MANCHESTER 1824

Phong interpolation

- Instead of interpolating colours, Phong suggested interpolating normal vectors
- We interpolate the normal vector along the scanline
- Compute illumination model for every pixel

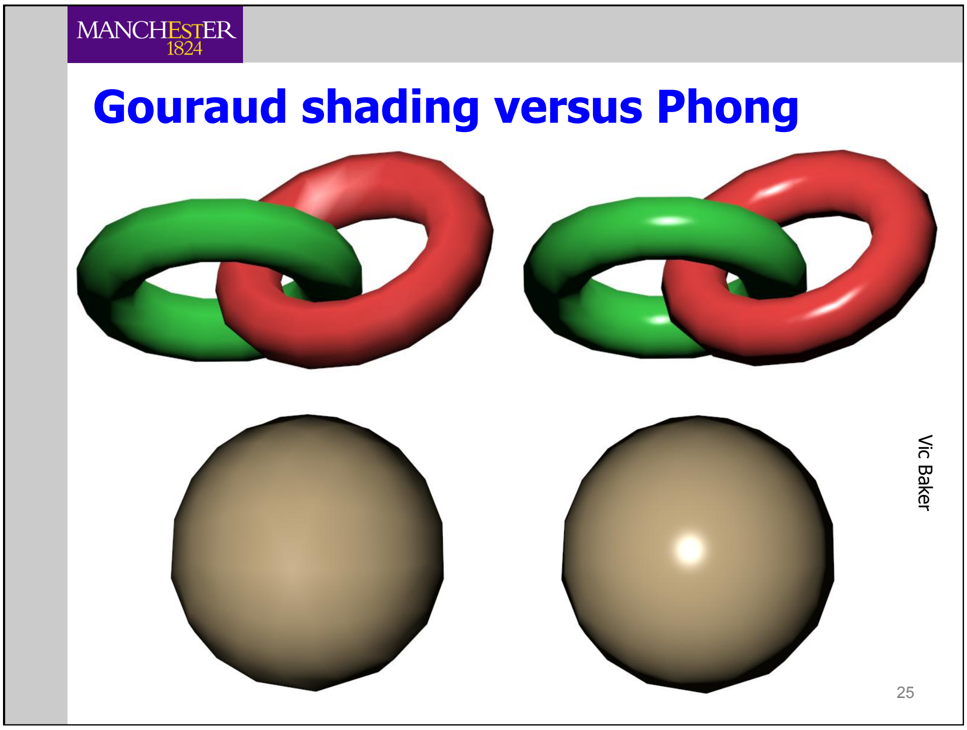


MANCHESTER 1824

Phong algorithm

- for each scanline {
  - compute average normal **N_{Left}** from **N_A** and **N_C**
  - compute average normal **N_{Right}** from **N_B** and **N_C**
  - average between **N_{Left}** and **N_{Right}** along the scanline, and compute colour C using illumination model

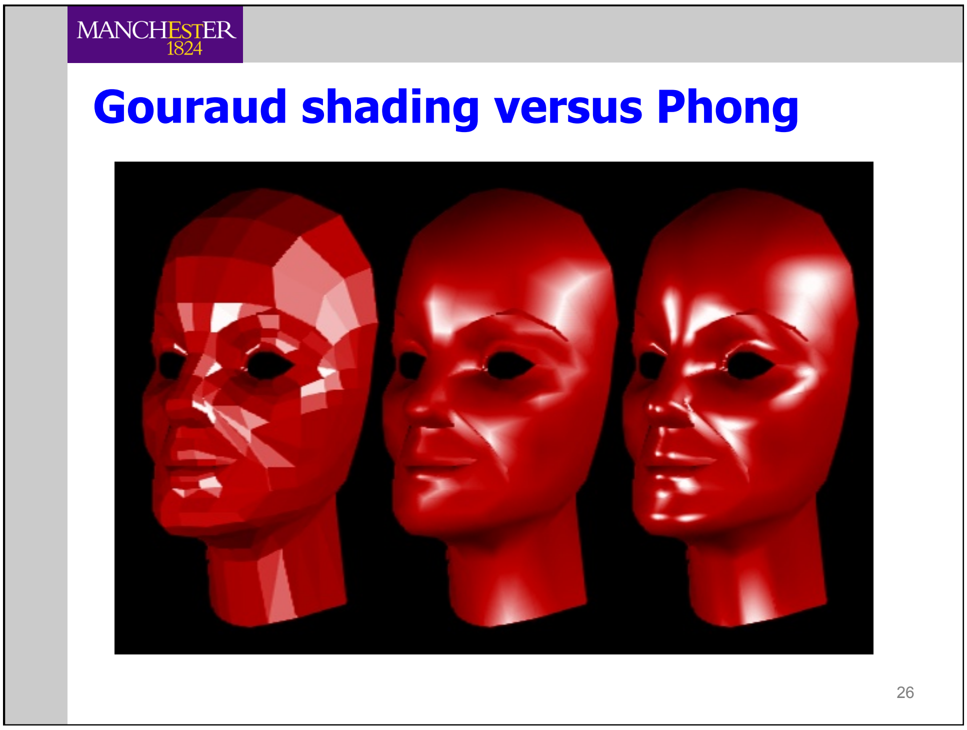
  }



Gouraud shading versus Phong

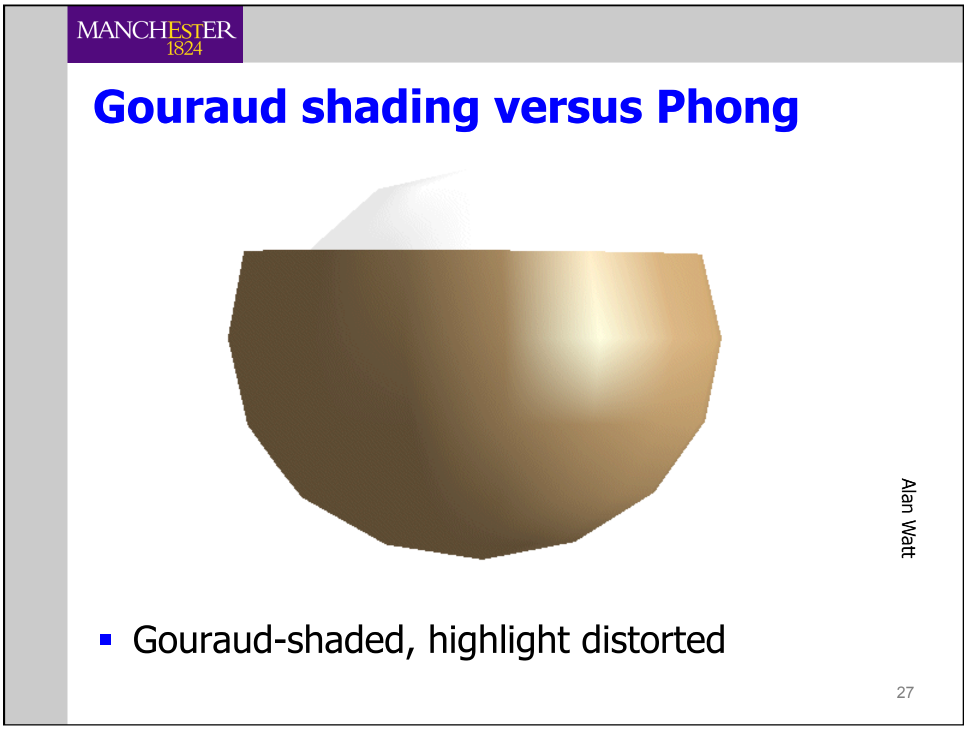
Vic Baker

25



Gouraud shading versus Phong

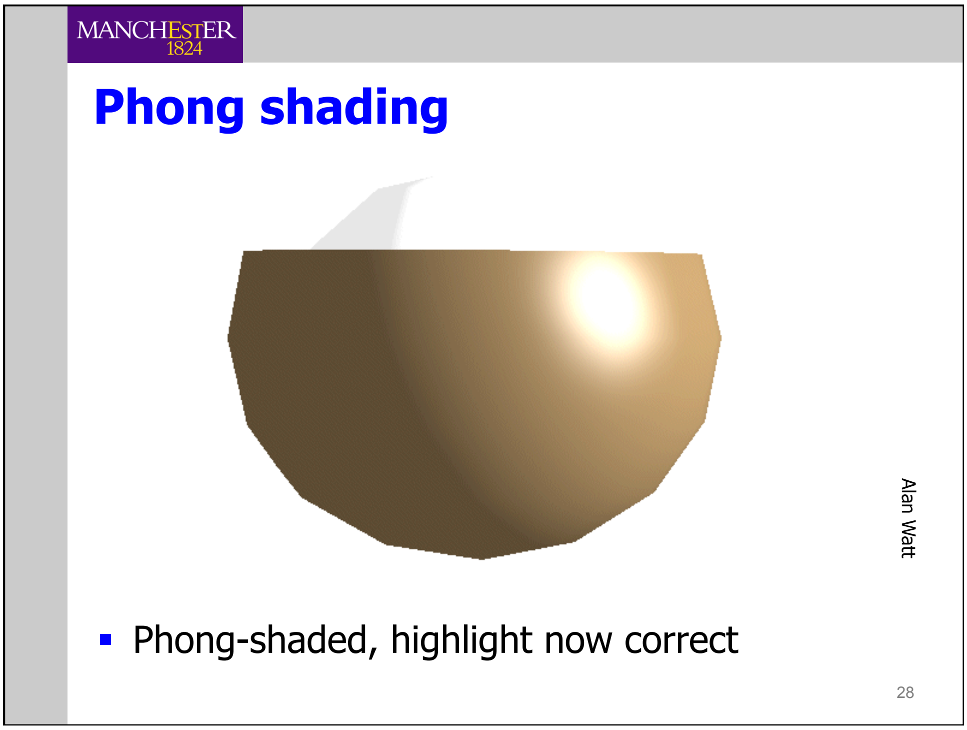
26



Gouraud shading versus Phong

Alan Watt

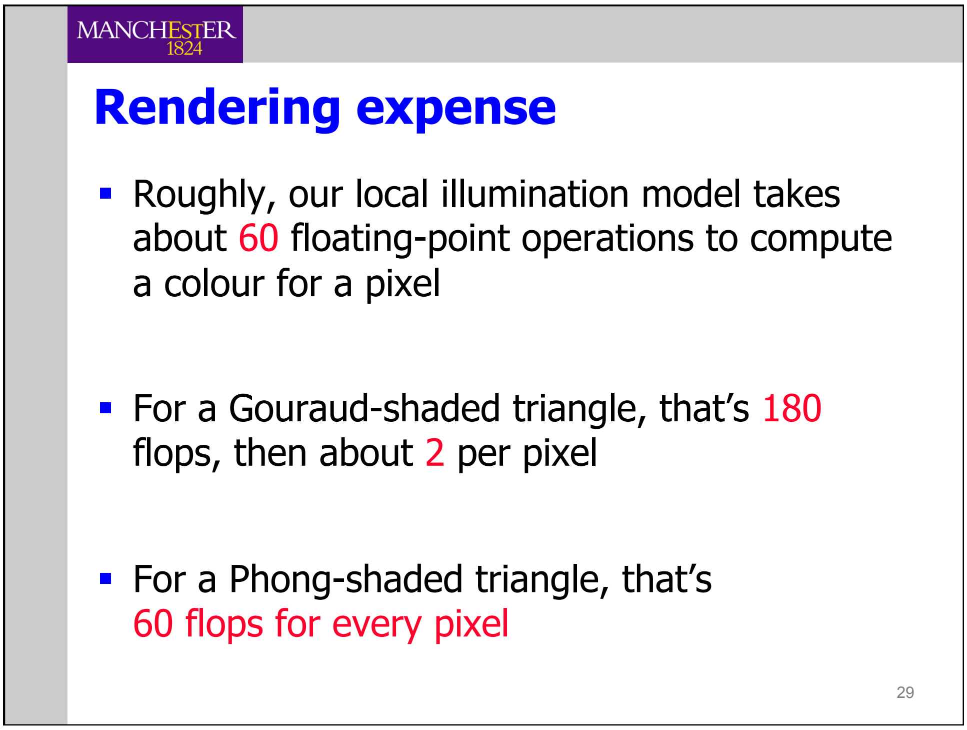
- Gouraud-shaded, highlight distorted



Phong shading

Alan Watt

- Phong-shaded, highlight now correct



Rendering expense

- Roughly, our local illumination model takes about 60 floating-point operations to compute a colour for a pixel
- For a Gouraud-shaded triangle, that's 180 flops, then about 2 per pixel
- For a Phong-shaded triangle, that's 60 flops for every pixel

29