

# Computer Graphics

P. Healy





CS1-08  
Computer Science Bldg.  
tel: 202727  
`patrick.healy@ul.ie`

Spring 2021–2022

# Outline

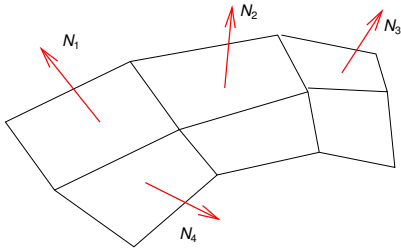
## 1 Polygon Rendering Methods; §17-10

# Introduction

- Through our *illumination model* we now have the ability to calculate the light intensity at any  in a ;
- This can be very expensive so a possibility is to assume a  light intensity throughout each of the polygons that approximates our surface;
- Alternatively we can use this model to calculate the intensity at selected points and interpolate (approximate) the  at other positions;
- Since *scan lines* are so central to the rendering process, graphics packages usually use the intersection points of scan lines and a polygon as locations to calculate light intensity;
- Other points on the scan line are then linearly interpolated





# Introduction

- Through our *illumination model* we now have the ability to calculate the light intensity at any  in a ;
- This can be very expensive so a possibility is to assume a  light intensity throughout each of the polygons that approximates our surface;







- Alternatively we can use this model to calculate the intensity at selected points and interpolate (approximate) the  at other positions;
- Since *scan lines* are so central to the rendering process,





# Introduction

- Through our *illumination model* we now have the ability to calculate the light intensity at any  in a ;
- This can be very expensive so a possibility is to assume a  light intensity throughout each of the polygons that approximates our surface;
- Alternatively we can use this model to calculate the intensity at selected points and interpolate (approximate) the  at other positions; we will see two different techniques that use this idea later.
- Since *scan lines* are so central to the rendering process, graphics packages usually use the intersection points of scan lines and a polygon as locations to calculate light intensity;
- Other points on the scan line are then linearly interpolated





# Introduction

- Through our *illumination model* we now have the ability to calculate the light intensity at any  in a ;
- This can be very expensive so a possibility is to assume a  light intensity throughout each of the polygons that approximates our surface;
- Alternatively we can use this model to calculate the intensity at selected points and interpolate (approximate) the  at other positions;
- Since *scan lines* are so central to the rendering process, graphics packages usually use the intersection points of scan lines and a polygon as locations to calculate light intensity;
- Other points on the scan line are then linearly interpolated

# Introduction

- Through our *illumination model* we now have the ability to calculate the light intensity at any  in a ;
- This can be very expensive so a possibility is to assume a  light intensity throughout each of the polygons that approximates our surface;
- Alternatively we can use this model to calculate the intensity at selected points and interpolate (approximate) the  at other positions;
- Since *scan lines* are so central to the rendering process, graphics packages usually use the intersection points of scan lines and a polygon as locations to calculate light intensity;
- Other points on the scan line are then linearly interpolated

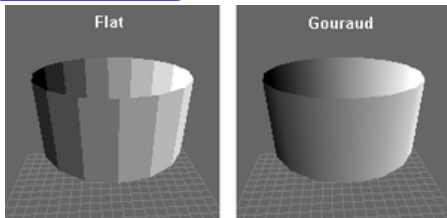
# Constant-Intensity Surface Rendering

- Fastest, simplest method for rendering a polygon
- Also known as **flat surface rendering**
- Method works when
  - Polygon is one  of a polyhedron (has sharp corners) and is not a section of a curved-surface approximation mesh
  - All light sources illuminating the polygon are  away from surface that  $N \cdot L$  and attenuation function are both constant over area of poly (diffuse)
  -  is far enough away from poly so that  $V \cdot R$  is  over area of poly (specular)
- Although we usually want to *hide* the polygons that approximate a surface, for debugging a design that *exposes* the approximation may be useful



# Gouraud Surface Rendering

- One problem with flat surface rendering is that [redacted] arise on boundaries of polygons
- A better algorithm, though still not perfect, is due to Henri Gouraud
- Idea is to calculate [redacted] at polygon vertices
- Then, linearly interpolate these intensities across polygon faces of a lighted object
- Eliminates [redacted] that occur with flat shading



Comparison of Flat Shading and Gouraud Shading

- See [here](#) for more details

# Gouraud Surface Rendering (contd.)

## • Procedure

- Determine average unit normal vector at each vertex of poly For a vertex  $v$  that is the meeting point for  $n$  polygons, each with normal  $\mathbf{N}_k$  compute the “average” normal

$$\mathbf{N}_v = \frac{\sum_{k=1}^n \mathbf{N}_k}{|\sum_{k=1}^n \mathbf{N}_k|}$$

Why no  $n$  in denominator?

Also, some people argue that not all the normals should be considered as having an equal contribution: if  $N_k$  is of a very small polygon then it should be weighted smaller in the sum. Weight can be 1) relative area of each polygon or 2) relative angle of the two polygon sides at vertex  $v$ .

- Apply illumination model at each vertex to obtain the light intensity at that position
- Linearly interpolate the vertex intensities using scan lines over projected area of polygon

# Gouraud Surface Rendering (contd.)

- Procedure

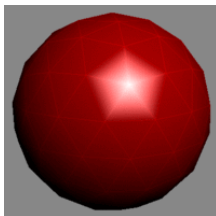
- Determine average unit normal vector at each vertex of poly
- Apply illumination model at each vertex to obtain the light intensity at that position
- Linearly interpolate the vertex intensities using scan lines over projected area of polygon

# Gouraud Surface Rendering (contd.)

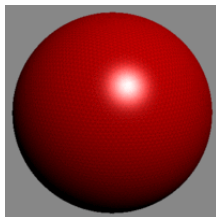
- Procedure

- Determine average unit normal vector at each vertex of poly
- Apply illumination model at each vertex to obtain the light intensity at that position
- Linearly interpolate the vertex intensities using scan lines over projected area of polygon

# Gouraud Surface Rendering (contd.)

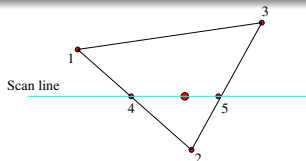


Gouraud shading with low polygon count



Gouraud shading with high polygon count

# Scan Line Linear Interpolation



- Having calculated intensities at all vertices ( $I_1, I_2, I_3$ ) we calc intensities along scan lines
- $I_4$  intensity can be calculated with

$$I_4 = \frac{y_4 - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y_4}{y_1 - y_2} I_2$$

and likewise for  $I_5$

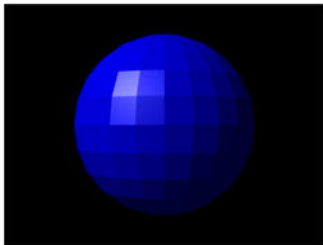
- A pixel  $p$  between points 4 and 5 has intensity

$$I_p = \frac{x_5 - x_p}{x_5 - x_4} I_4 + \frac{x_p - x_4}{x_5 - x_4} I_5$$

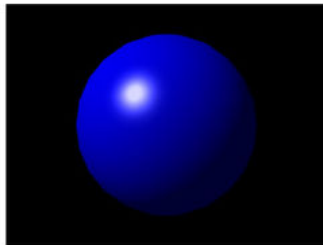
- If scan lines are treated successively then further simplifications are possible

# Phong Surface Rendering

- Best known shading algorithm is Phong shading
- Idea here is to interpolate the normal vectors instead of the light intensity
- Provides more accurate calculation of light intensity values and more realistic surface highlights



FLAT SHADING



PHONG SHADING