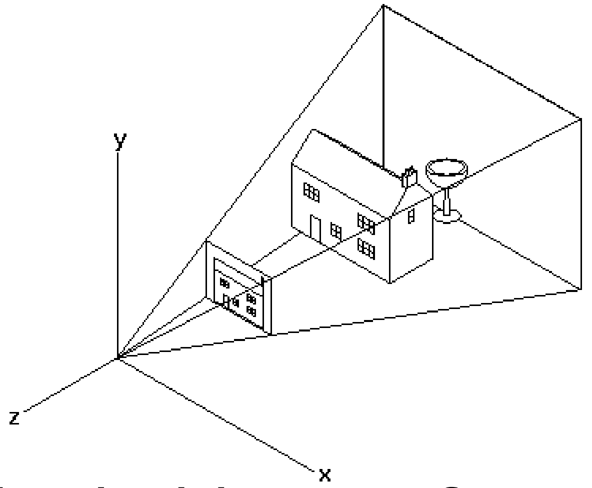
CS5182 Computer Graphics
Hidden Surface Removal
Illumination & Shading

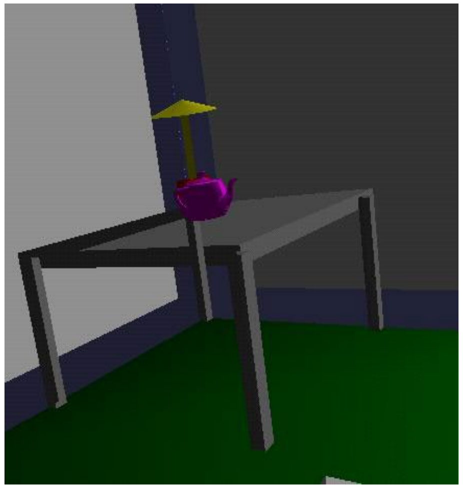
2024/25 Semester A

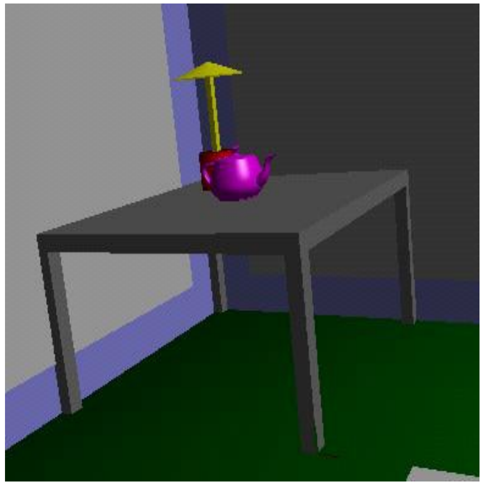
City University of Hong Kong (DG)

Hidden Surface Removal

- What is hidden surface removal?
  - Visibility algorithm or hidden surface removal is concerned with determining what is visible within a scene from a chosen viewing position.




Wrong visibility


Correct visibility

- Why hidden surface removal?
  - A correct rendering requires correct visibility calculations. Correct visibility---when multiple opaque polygons cover the same screen space, only the front most one is visible (remove the hidden surfaces).

Hidden Surface Removal

- Object space algorithms
  - Determine which 3D objects are in front of others. They do their work on the 3D objects themselves before they are converted to pixels in the frame buffer.

  -- Back-face culling algorithm

- Image space algorithms
  - Determine what color is used to paint at each pixel. They do their work as the objects are being converted to pixels in the frame buffer. The test is carried out in 2D space.

  -- Depth-sorting (or Painter's) algorithm

  -- Z (or Depth)-buffering algorithm

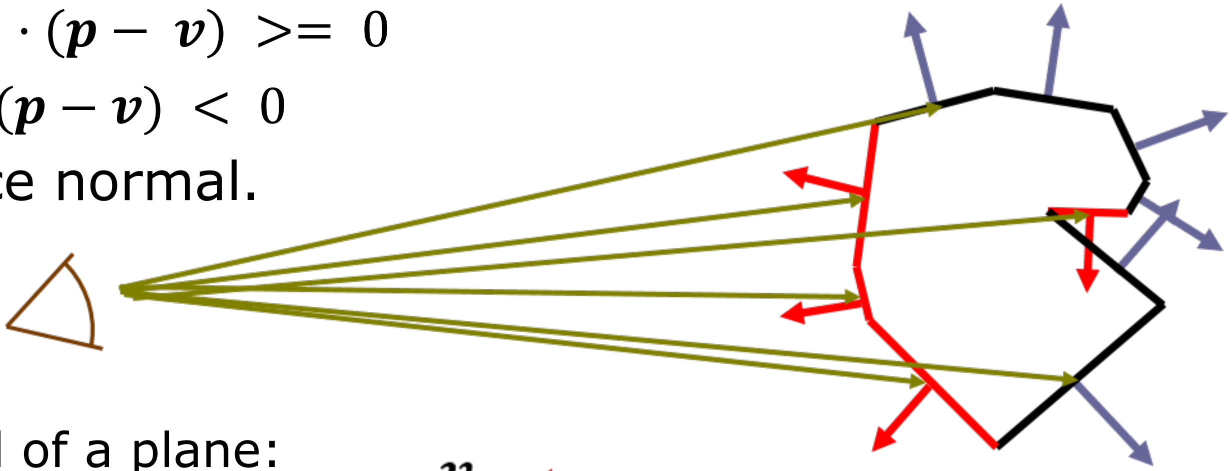
Hidden Surface Removal

- Back-face culling algorithm
  - 3D objects are usually made of polygons. Draw only the polygons facing the camera and hide the ones which are facing away from viewpoint.
  - For each polygonal face f, compute the vector from viewpoint $\boldsymbol{v}$ to any point $\boldsymbol{p}$ on f

    - Invisible if $\boldsymbol{n} \cdot (\boldsymbol{p} - \boldsymbol{v}) >= 0$

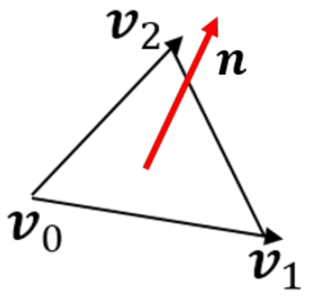
    - Visible if $\boldsymbol{n} \cdot (\boldsymbol{p} - \boldsymbol{v}) < 0$

  where $\boldsymbol{n}$ is the face normal.



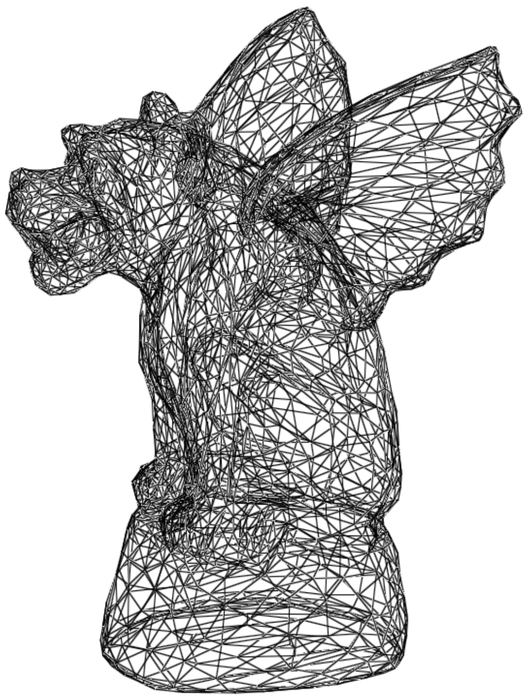
Calculate the normal of a plane:

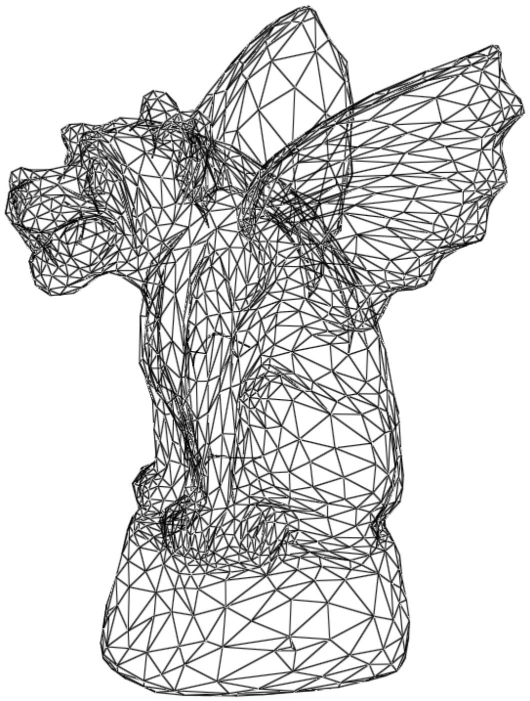
$$\boldsymbol{n} = (\boldsymbol{v}_1 - \boldsymbol{v}_0) \times (\boldsymbol{v}_2 - \boldsymbol{v}_0)$$



Hidden Surface Removal

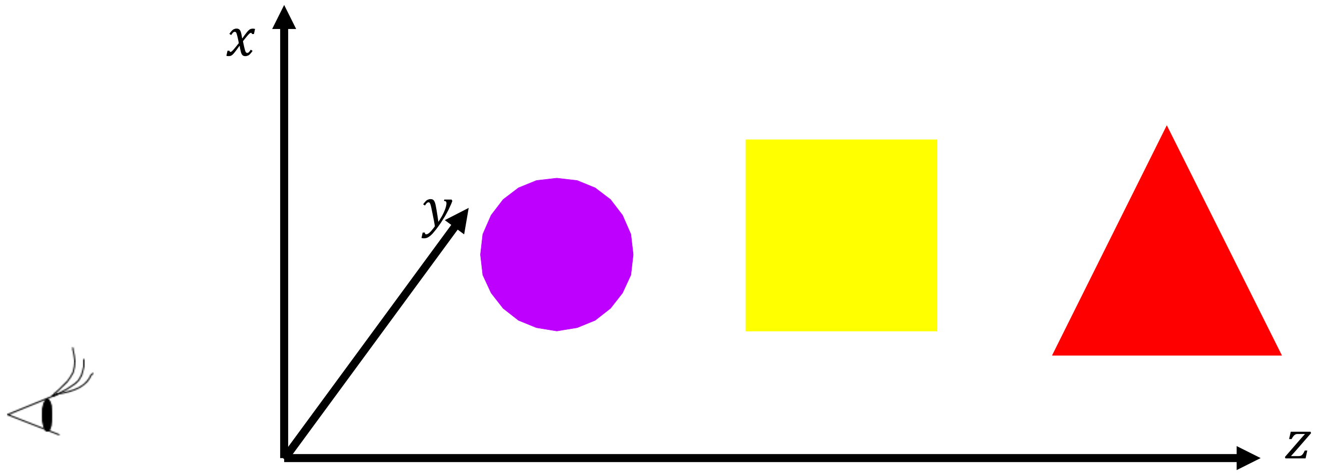
- Back-face culling algorithm
  - Advantage: easy to implement and highly efficient; the complexity is O(*n*) where *n* is the number of polygons.
  - Disadvantage: fail for concave polyhedral.

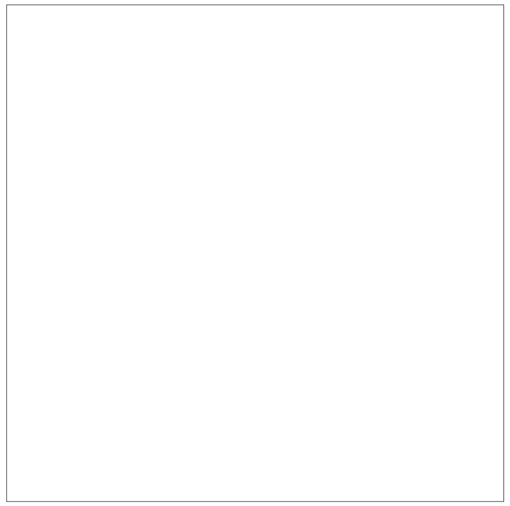

Drawing all triangles


With back face culling

Hidden Surface Removal

- Depth-sorting (or Painter's) algorithm

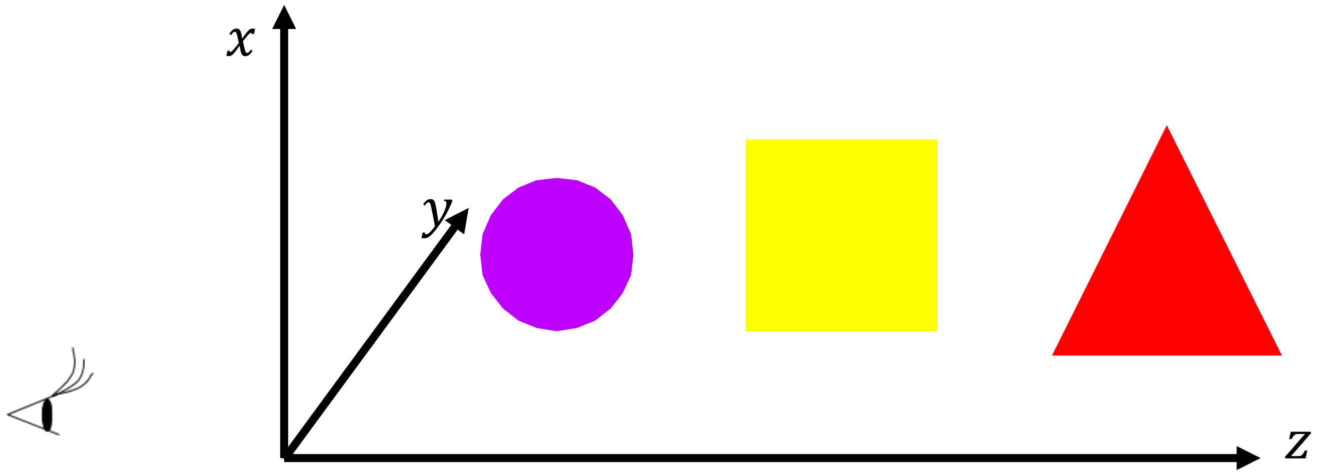


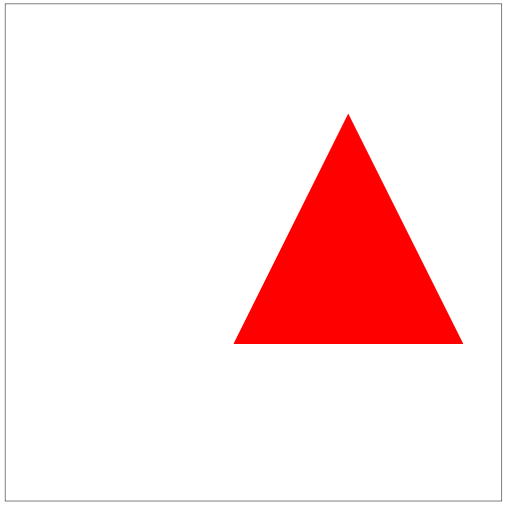


Frame buffer

Hidden Surface Removal

- Depth-sorting (or Painter’s) algorithm

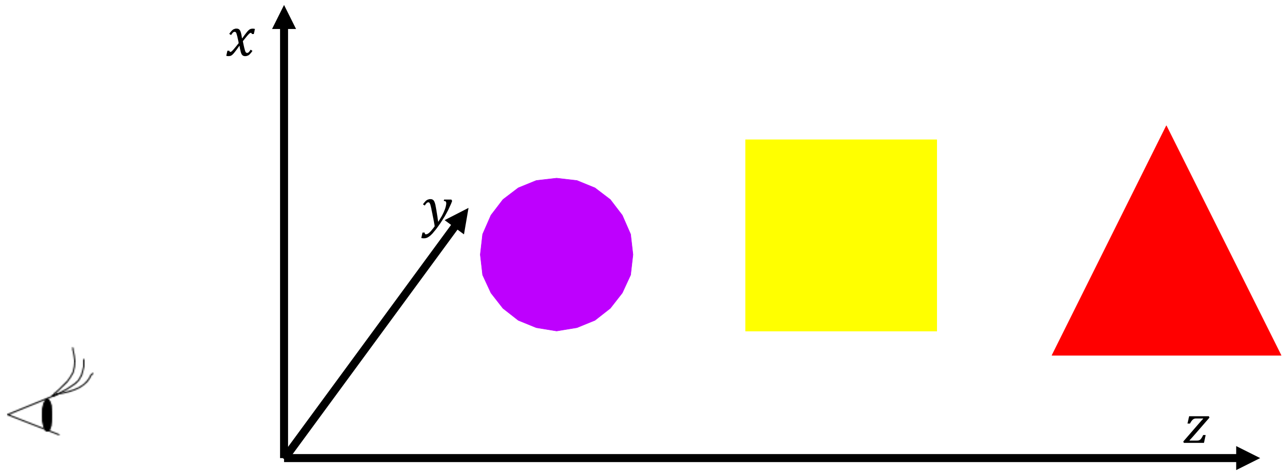


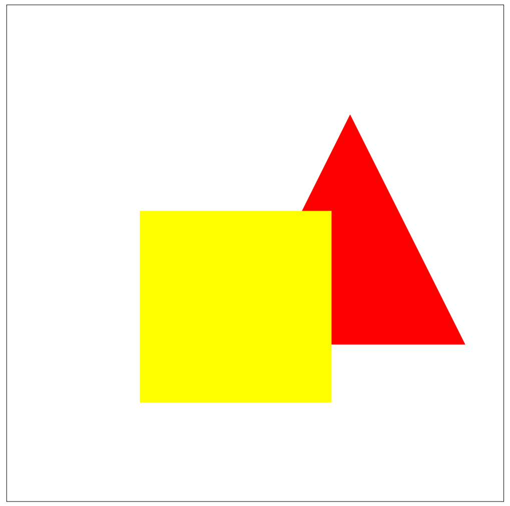


Frame buffer

Hidden Surface Removal

- Depth-sorting (or painter's) algorithm

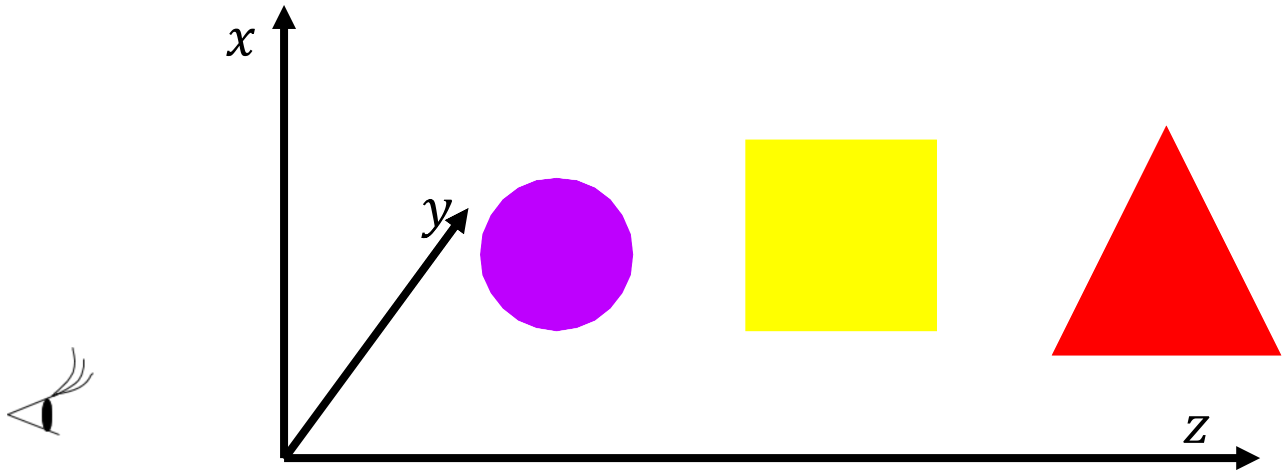




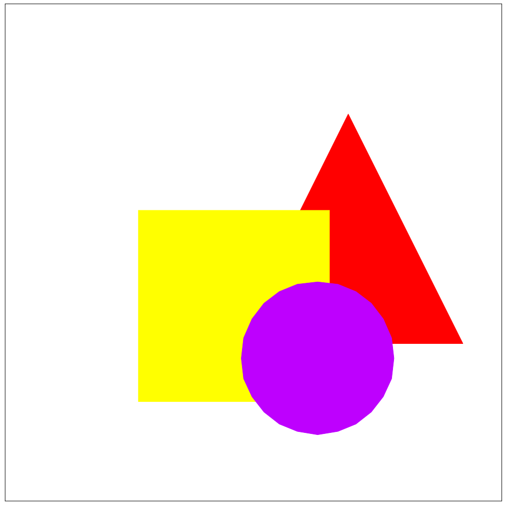
Frame buffer

Hidden Surface Removal

- Depth-sorting (or painter's) algorithm



Final image



Hidden Surface Removal

- Depth-sorting (or Painter's) algorithm
  - An algorithm for creating a hidden-line drawing of polygon data sets by drawing the polygons from the most distant to the closest, in order.

1) Sort all surfaces according to their distances from the view point.

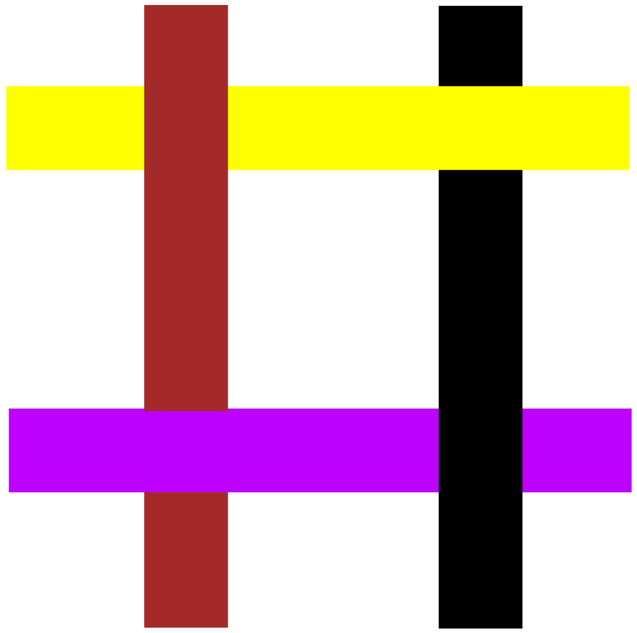
2) Render the surfaces to the frame buffer one at a time starting from the ***farthest*** surface.

3) A surface drawn later will replace overlapping surfaces drawn earlier.

4) After all the surfaces have been processed, the frame buffer stores the final image.

Hidden Surface Removal

- Depth-sorting (or Painter’s) algorithm
  - The basic idea of this method is simple. When there are only a few objects in the scene, this method can be very efficient. However, as the number of objects increases, the sorting process can become very complex and time consuming.
  - Certain conditions, such as intersecting polygons or cyclic overlap are not processed correctly.

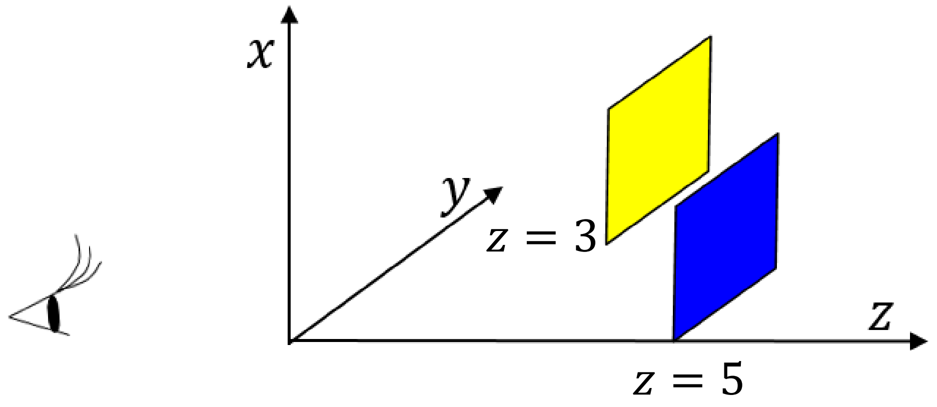


Hidden Surface Removal

- Z (or Depth)-buffering algorithm
  - This method requires two buffers, a *frame buffer* and a *z-buffer* (or depth buffer).
  - The depth buffer has the same resolution as the frame buffer and is used to store the depth values (distances from the view point) of the visible surfaces. It is also used to determine the nearest surface at each pixel.

Hidden Surface Removal

- Z (or Depth)-buffering algorithm
  - Initialize the depth buffer and frame buffer to very large positive values and the background color, respectively



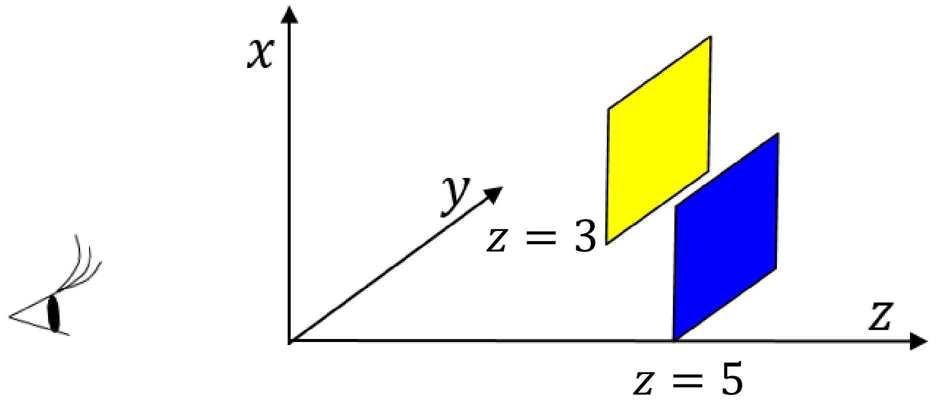
999	999	999	999
999	999	999	999
999	999	999	999
999	999	999	999

Depth buffer

Frame buffer

Hidden Surface Removal

- Z (or Depth)-buffering algorithm
  - Draw the blue polygon (the order does not affect the final result anyway).



999	999	999	999
999	999	999	999
5	5	999	999
5	5	999	999

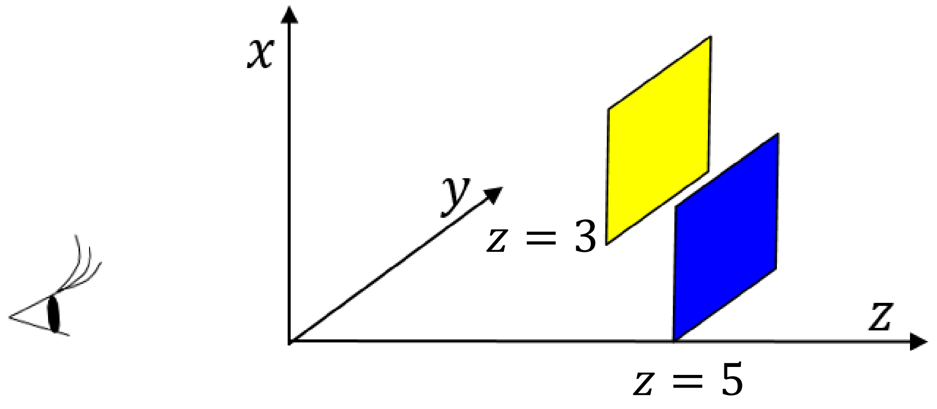
Depth buffer

🟦	🟦		
🟦	🟦		

Frame buffer

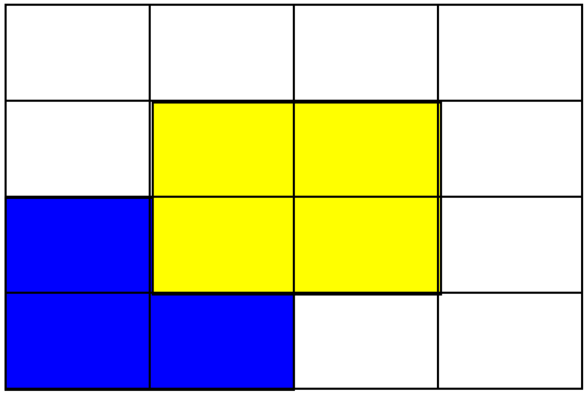
Hidden Surface Removal

- Z (or Depth)-buffering algorithm
  - Then draw the yellow polygon



999	999	999	999
999	3	3	999
5	3	3	999
5	5	999	999

Depth buffer



Frame buffer

Hidden Surface Removal

- Z (or Depth)-buffering algorithm

  1) Initially, each pixel of the depth buffer is set to the maximum depth value (the depth of the back clipping plane), and the frame buffer is set to the background color.

  2) Surfaces are rendered one at a time at random.

  3) The depth value of each pixel of a surface is calculated.

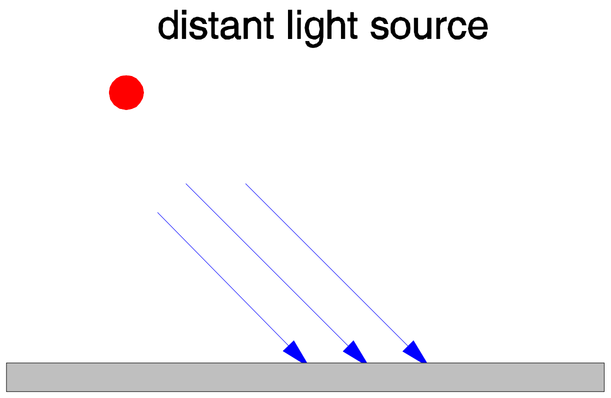
  4) If this depth value is smaller than the corresponding depth value in the depth buffer (i.e., it is closer to the view point), both the depth value in the depth buffer and the color value in the frame buffer are replaced by the depth value and the color value of this surface.

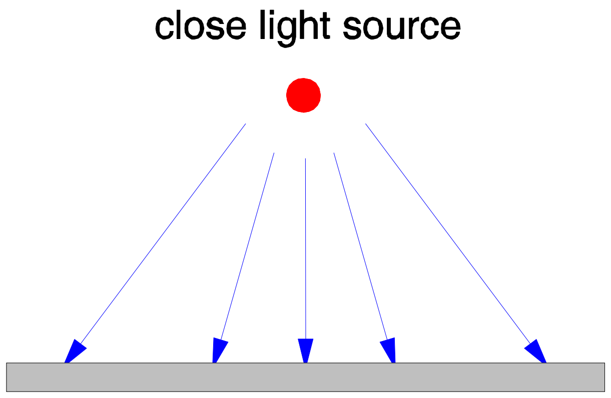
Hidden Surface Removal

- Z (or Depth)-buffering algorithm
  - Because of the need for an addition buffer and the overheads involved in updating the buffer, this method is less attractive in comparison with the depth-sorting method for applications with only a few objects in the scene.
  - However, as the number of objects in the scene increases, the overheads become less important and the method becomes more attractive.
  - With the significant reduction in memory cost, the cost for the additional depth buffer is no longer an issue.
  - Majority of graphics accelerators available in the market are based on the z-buffer method.

Illumination

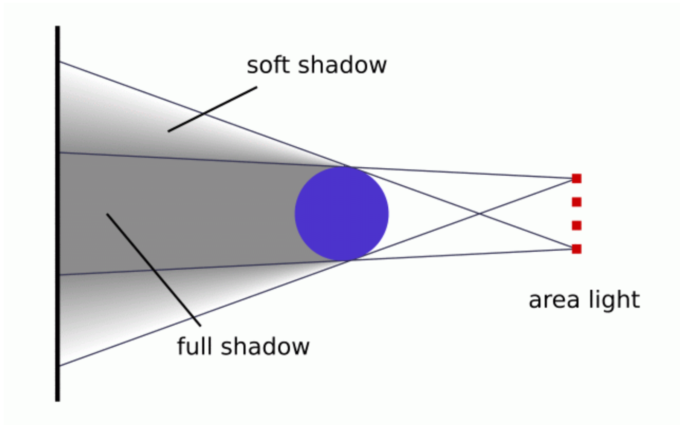
- Light sources
  - *Ambient light*: represents the environment or background light. We usually approximate it with a uniform intensity from all directions, causing objects to be uniformly illuminated from any viewing position.
  - *A point light source*: represents an ideal light source. It emits directional light rays. Normally, light rays from a distant light source (e.g., sun light) may be considered as parallel and those from a close light source (e.g., table lamp) as divergent.





Illumination

- Light sources
  - *Spotlight* (a.k.a. searchlight): projects a powerful beam of light of approximately parallel rays in a particular direction.
  - *An Area light source*: occupies a finite, one- or two-dimensional area of space. It can cast soft shadows because an object can partially block their light.



lightcuts - textured area light - 7198 lights
Rendering 08:49.74

Illumination Model

- An ***illumination model*** (aka a lighting model) determines the color of a surface point by simulating some light attributes, i.e., the flux of light energy from light sources to objects in the scene via direct and indirect paths.

  - The *Phong illumination model* consists of three parts:

  -- Ambient reflection

  -- Diffuse reflection

  -- Specular refection

Phong Illumination Model

- Ambient reflection
  - Ambient light intensity I_c is equal in all directions.
  - For a given surface, we can specify the amount of background light that the surface reflects I_a

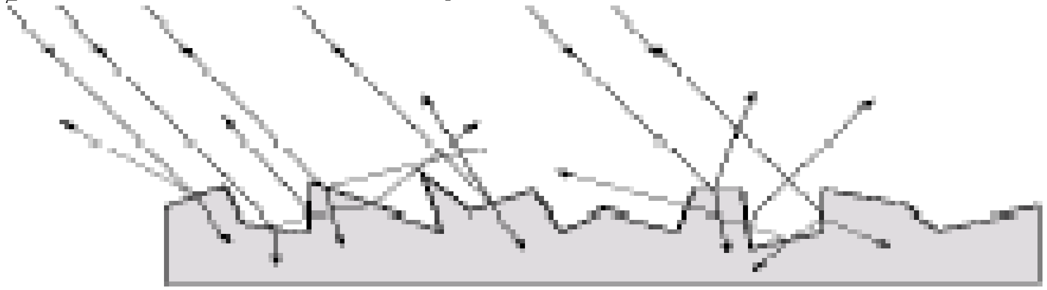
$$I_a = k_a * I_c$$

where k_a is an ambient reflection coefficient and $0 < k_a < 1$.

  - Determined by the surface properties and independent of the surface’s position and orientation.

Phong Illumination Model

- Diffuse reflection
  - Because of the microscopic variations, an incoming ray of light is **equally** reflected in any direction over the hemisphere.



  - [Lambert's Law] The reflected intensity at each point is proportional to $\cos\theta$:

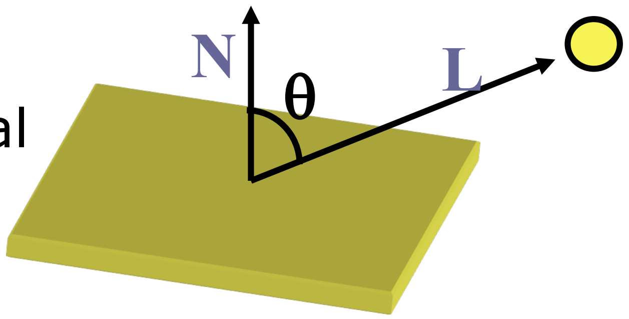
$$I_d = k_d * I_L * \cos\theta = k_d * I_L * (\mathbf{N} \cdot \mathbf{L})$$

where

k_d – surface diffuse coefficient.

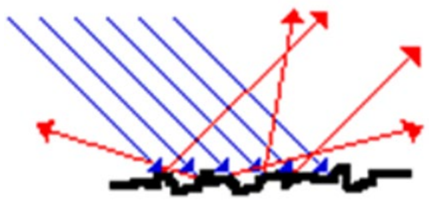
θ - the angle between the surface normal and the incoming light ray, called the angle of incidence.

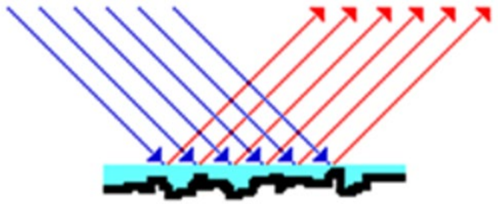
N and **L** are normalized vectors.



Phong Illumination Model

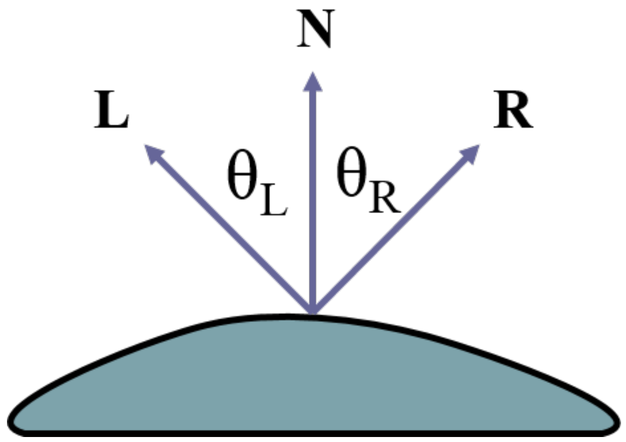
- Specular reflection
  - Account for the bright spots on shiny, glossy surfaces (like metal, plastics).
  - Specularities are caused by microscopically smooth surfaces.
  - Shiny surfaces change appearance when viewpoint is changed.

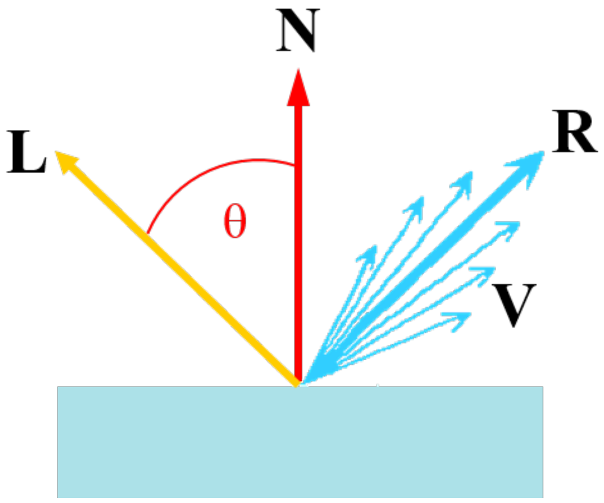

A dry asphalt roadway diffuses incident light.


When wet, water fills in the crevices, resulting in specular reflection and a glare.

Phong Illumination Model

- Specular reflection
  - An *ideal* specular surface (perfect mirror reflector) is very smooth at the microscopic level. For an ideal specular surface, the incoming ray **L**, the surface normal **N**, and the reflected ray **R** lie in a common plane and $\theta_L = \theta_R$
  - For a *near-perfect* reflector, you might expect the highlight to fall off quickly with the viewing direction **V** deviating from the ideal reflected direction **R**.





Phong Illumination Model

- Specular reflection
  - Phong B.T. proposed the following model to calculate the specular reflection:

$$I_s = k_s * I_L * \cos^n \phi = k_s * I_L * (\mathbf{V} \cdot \mathbf{R})^n$$

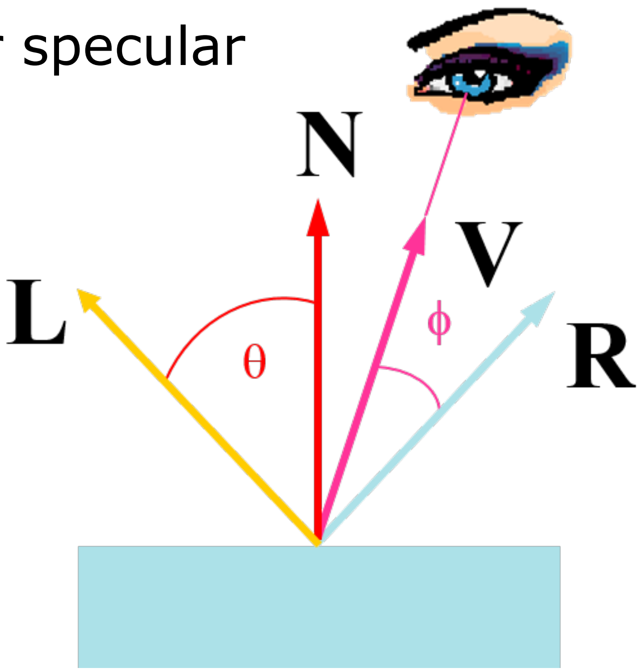
where

k_s – surface specular reflection coefficient.

n - specular-reflection parameter or specular exponent (for mirror $n = \infty$).

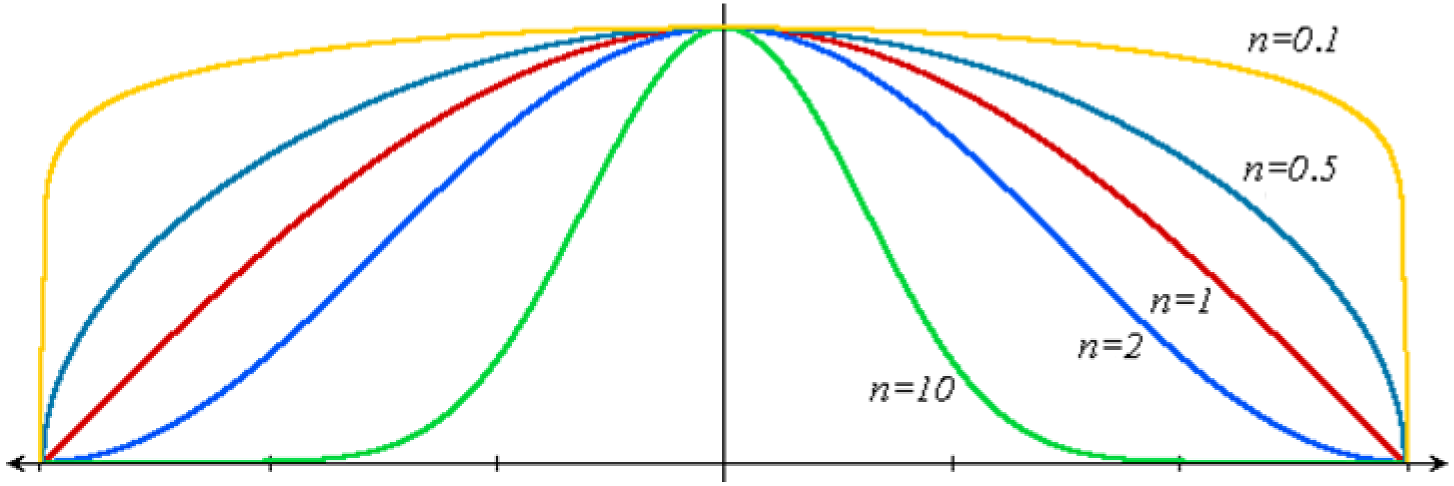
ϕ – angle between **R** and **V**

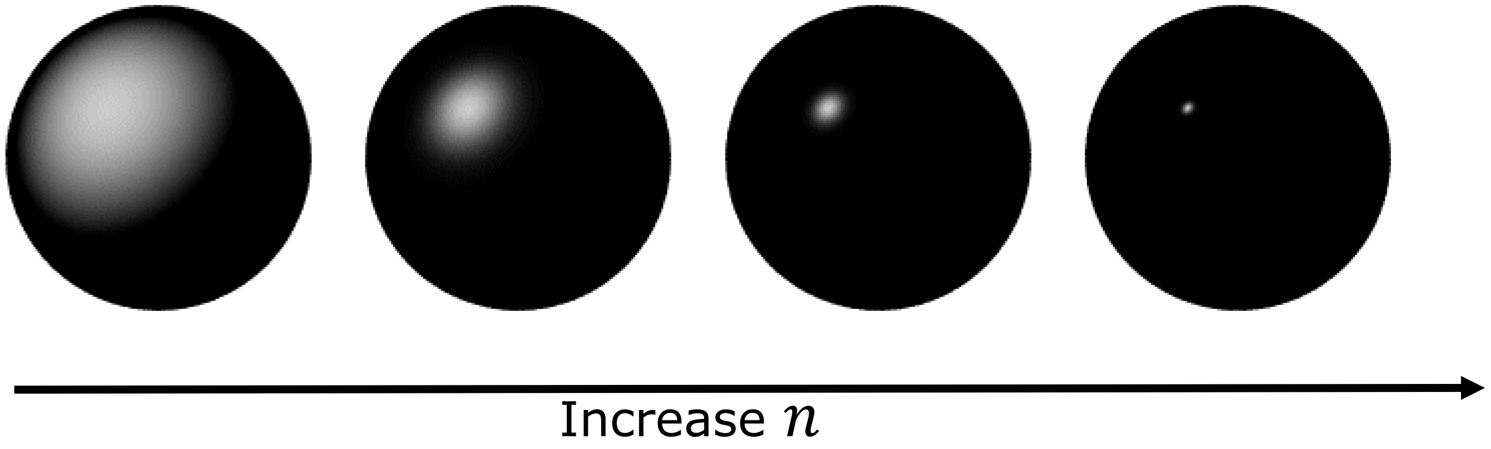
R and **V** are normalized vectors.



Phong Illumination Model

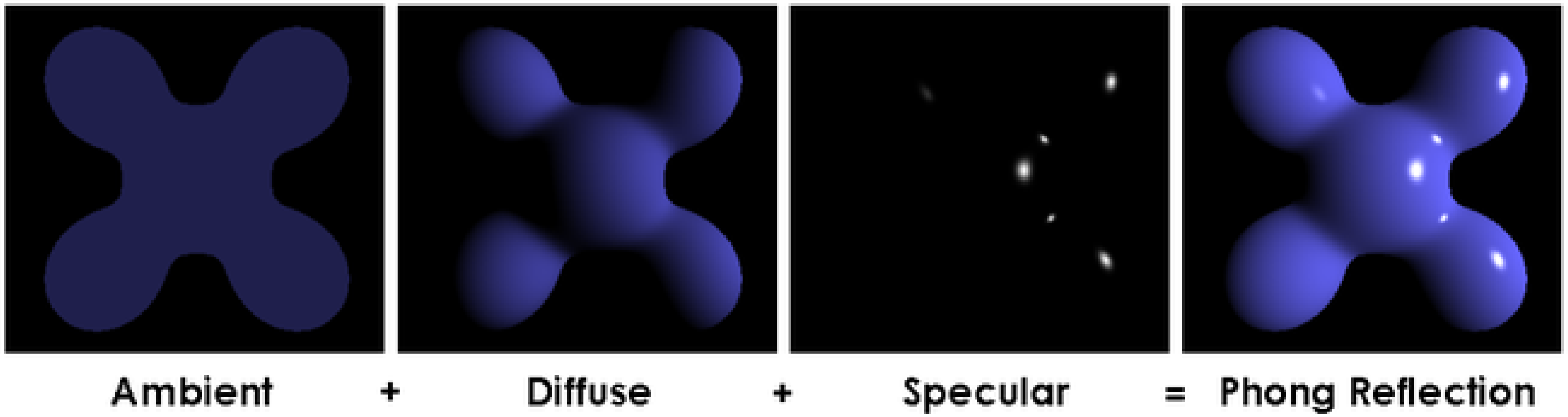
- Specular reflection $I_s = k_s * I_L * \cos^n \phi$
  - The exponent *n* controls the rate of the fall-off.





Phong Illumination Model

$$I = I_a + \sum_{i=1}^{\#\,lights} (I_{d,i} + I_{s,i})$$



Shading

- A ***shading method*** (aka a surface rendering method) uses the color calculation from an illumination model to determine the pixel colors for all projected positions in a scene.

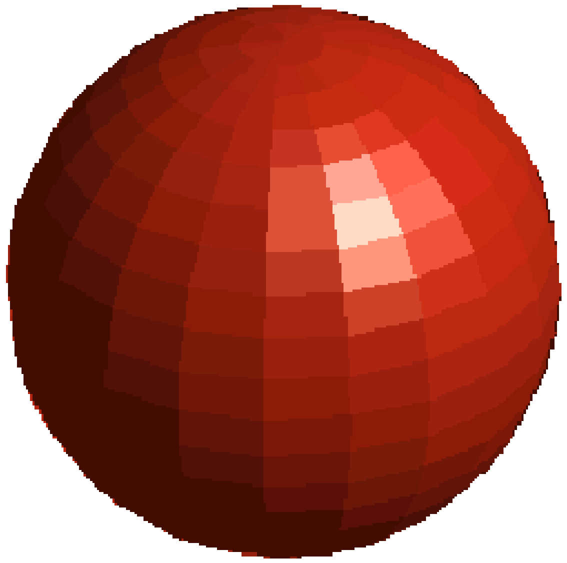
  - Flat shading
  - Smooth shading

  -- Gouraud shading

  -- Phong shading

Shading

- Flat shading
  - For each polygon, we only compute the intensity value once. The whole polygon will be shaded with the same intensity value.
  - This method is fast and simple but it can only produce very coarse reflection. Good for far away light and viewer or the surface is quite small (close to pixel size).
  - The major problem of this method is intensity discontinuity between adjacent polygons.



Shading

- Smooth shading: need to have per-vertex normals
  - Gouraud shading

    -- Interpolate color across triangles

    -- Fast, supported by most of the graphics accelerator cards

  - Phong shading

    -- Interpolate normals across triangles

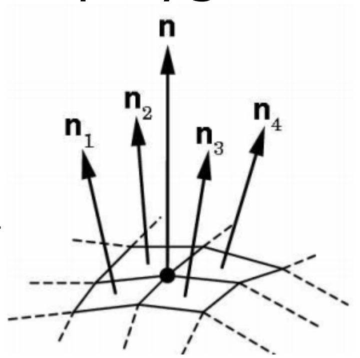
    -- More accurate, but slower than Gouraud shading.

Shading

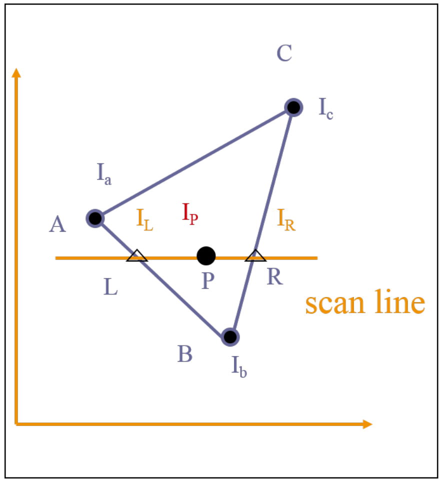
- Gouraud shading (proposed by Henri Gouraud)
  - The intensity value is calculated once for each vertex of a polygon. The intensity values for the inside of the polygon are obtained by interpolating the vertex values.

-- first, the surface normal of each vertex could be estimated by averaging all adjacent face normal:

$$\mathbf{n} = \frac{\mathbf{n}_1 + \mathbf{n}_2 + \mathbf{n}_3 + \mathbf{n}_4}{\left\| \mathbf{n}_1 + \mathbf{n}_2 + \mathbf{n}_3 + \mathbf{n}_4 \right\|}$$



-- second, linearly interpolate the vertex intensities for the remaining points inside of the polygon:



1) Find the endpoints L, R

2) compute

$$\alpha = \frac{|AL|}{|AB|}, \quad \beta = \frac{|CR|}{|CB|}, \quad \gamma = \frac{|LP|}{|LR|}$$

3) compute

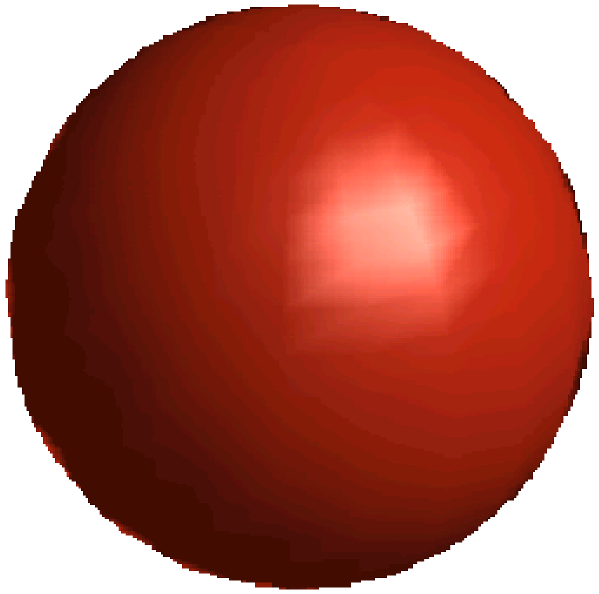
$$I_L = (1-\alpha)I_a + \alpha I_b$$
$$I_R = (1-\beta)I_c + \beta I_b$$

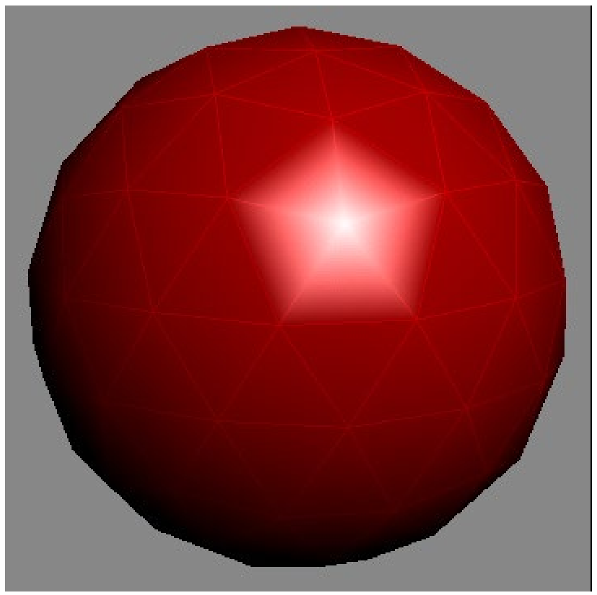
4) compute

$$I_p = (1-\gamma)I_L + \gamma I_R$$

Shading

- Gouraud shading (proposed by Henri Gouraud)
  - Gouraud shading does not properly handle specular highlights, specially when the n parameter is large (small highlight), since colors are linearly interpolated but specular result is non-linear.





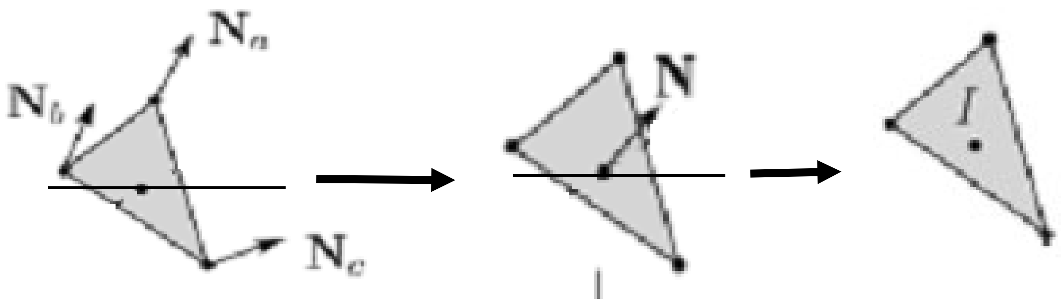
Shading

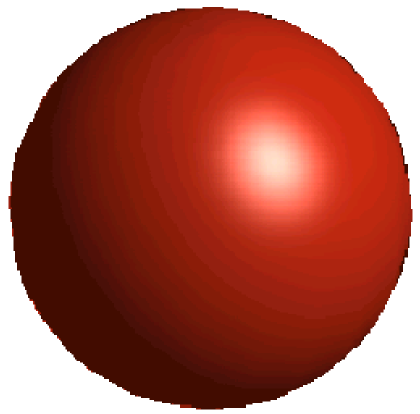
- Phong shading (proposed by Bui-Tuong Phong)
  - Instead of interpolating the intensity values, the normal vectors are being interpolated between the vertices. The intensity value is then calculated at each pixel using the light vector and the interpolated normal vector.

-- 1) Determine the unit normal at each polygon vertex.

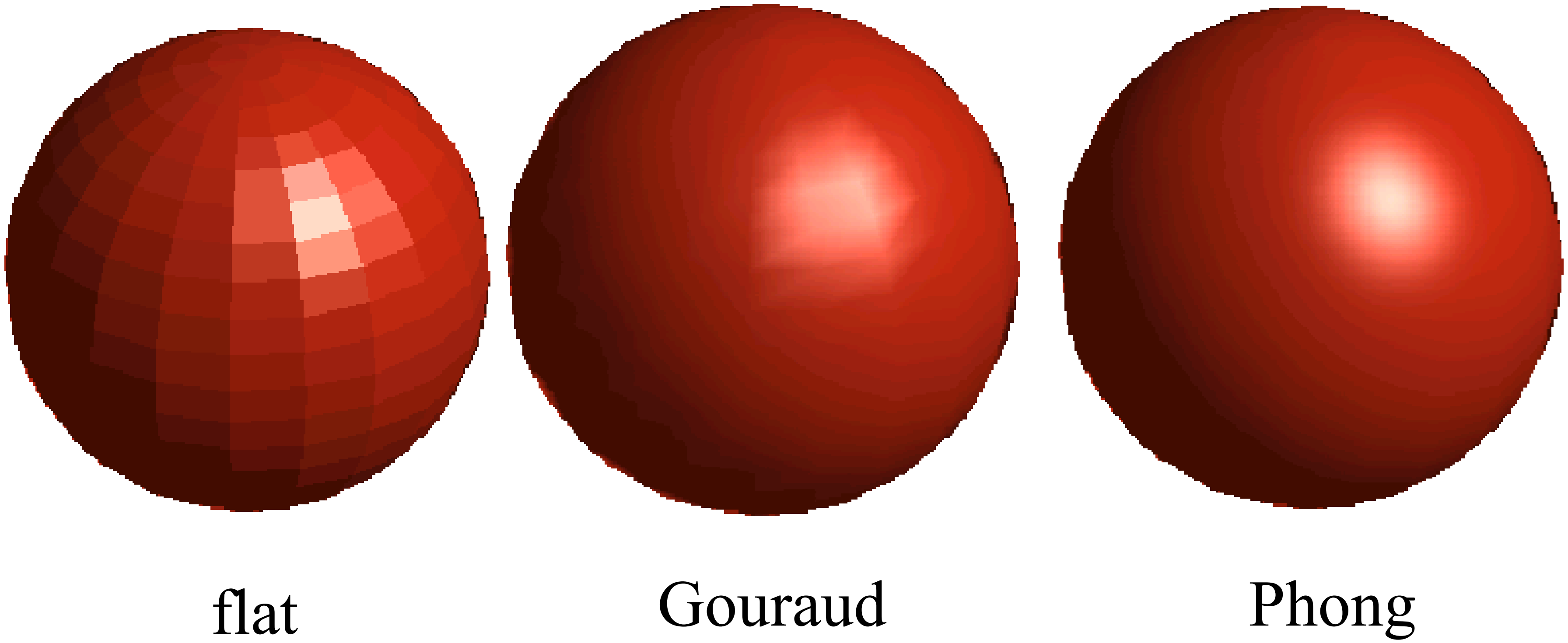
-- 2) Linearly interpolate the vertex normals over the surface polygon using barycentric coordinates, and normalize.

-- 3) Apply the illumination model along each scan line to calculate pixel intensities for each surface point.





Comparison


flat
Gouraud
Phong