CS7.302
Basics of Computer Graphics
Module: Lighting and Shading

Avinash Sharma

Spring 2021

Lighting/Shading

- We know which pixels of the frame buffer belongs to which object after visibility and scan conversion.
- What colour to give to the pixel? “Current colour”??
- Depends on: the colour of the object, the material properties of the object, the colour of the light source, the angle of viewing with respect to object/lights, etc.
- **Lighting** and **shading**: Finding the colours for each pixel, perhaps after finding it on the extrema of the primitives.
- What is our guide? **Physics!**
- Computational Process:

  Abstract – Represent – Process

Different Terms

- **Illumination Model:** How to “light” an object point given its material properties, the light sources, and the camera?
- **Shading Model:** How the illumination model applies to objects such as polygons and points.
- **Lighting:** Different types of lights.
- **Shadows:** How are shadows cast by objects.

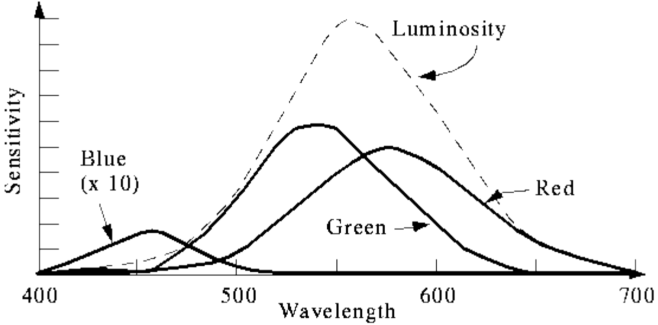
Colour Representation

- What do we write into the frame-buffer after identifying which pixels belong to a primitive?
- The **colour** and intensity!
- Colour represented using **Red**, **Green**, and **Blue**. Why?
- Visible portion of the electromagnetic spectrum:
  From about 400nm (**Violet**) to 700nm (**Red**).
- A bulb appears **red** if it emits light of λ in the red range
- A flower appears **orange** because it absorbs light of all wavelengths **except** in the orange range.

Human Colour Perception

- **Rods** of our retina see gray levels
- **Cones** are of 3 types: **Red**, **Green**, and **Blue**.
- Their spectral responses respectively peak approximately at 575 nm, 535nm, and 445nm
- *Luminous efficiency*: human visual sensitivity to constant luminance light
- Bell-shaped curve with a peak around 550nm (Green).

Human Eye Response to colour



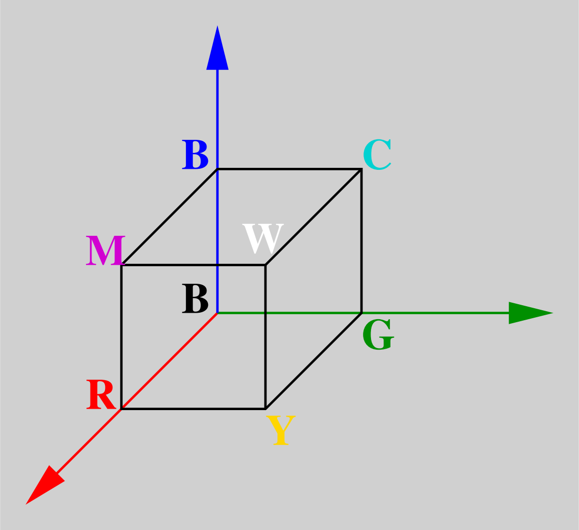
Tristimulus Theory

- All perceivable colours can be specified as weighted combination of primary colours R, G, and B.
- RGB define a set of basis vectors. Mix them in different measures to get any colour.
- Attractive, because a compact representation is possible.
- Problem reduces to: *Find RGB values to approximate a given perceptual colour*

RGB Colour Model

- Called the **Additive Primaries**.
- Colour Cube: R+G = **Yellow**, R+B = **Magenta**, G+B = **Cyan**, R+G+B = **White**, all zero gives **Black**.
- Popular in graphics, due to CRT/LCD Monitors
- Values commonly normalized to range $[0.0, 1.0]$. Externally 8-bits per colour channel
- Grays lie along the diagonal from (0, 0, 0) to (1, 1, 1).

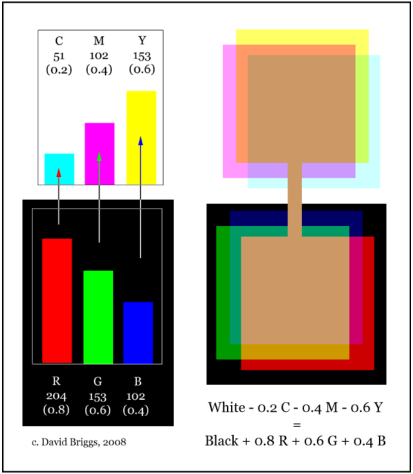
RGB Colour Model (cont.)



CMY(K) Colour Model

- Called the **Subtractive Primaries**.
- Colour Cube: C+M = **Blue**, C+Y = **Green**, M+Y = **Red**, C+M+Y = **Black**, all zero gives **White**!
- Subtract RGB from 1.0, you get CMY, respectively.
- Since black is common, $\min(C, M, Y)$ is taken out as black or K component. Yields the **CMYK** colour model used in the 4-colour printing process.

CMY(K) Colour Model



Different Terms

- **Illumination Model:** How to "light" an object point given its material properties, the light sources, and the camera?
- **Shading Model:** How the illumination model applies to objects such as polygons and points.
- **Lighting:** Different types of lights.
- **Shadows:** How are shadows cast by objects.

Illumination Models

- Modelling of the interaction with light and an object point from the point of view of the camera image.
- Three factors come into play: light source properties, material properties, and atmospheric effects.
- Light sources emit light.
  Properties: Colour and Directionality.
- Materials interact with light differently.
  Properties: Reflectivity and Colour.
- Atmospheric effects: attenuation.

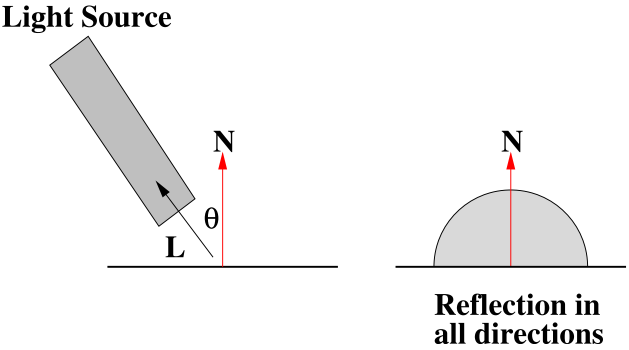
Material Properties

- **colour**: the paint used on the surface of the material with which the object is made.
- **Reflectivity**: how the material interacts with light.
  Shiny, dull, grainy, etc.
- Hard to separate these effects in practice.
  Combined impact is observed.
- **Diffuse reflection**: For dull, rough objects.
- **Specular reflection**: For smooth, shiny objects.

Diffuse or Lambertian Reflection

- Objects that obey Lambert's law of reflection: cloth, rough wall, etc.
- The normal component of light falling on it is absorbed by the object and is then reflected back equally in all directions.
- No preferred direction for reflecting light falling on it. Appearance of the point is independent of the view angle.
- Normal component of the light is proportional to $\cos\theta$.

Diffuse or Lambertian Reflection (cont.)



]

- Reflected light $\mathbf{I} \propto \mathbf{I_L} \cos\theta$ in all viewing directions.

Diffuse Illumination Equation

- The formula for computing the intensity at a point.
- For diffuse reflection, if I_p is the light falling at the surface,

$$I_d = I_p \, k_d \; \cos\theta = I_p \, k_d \, (\mathbf{N} \cdot \mathbf{L})$$

- k_d is the diffuse reflection coefficient.
- θ should be between 0 and 90. Otherwise, the light has no effect. (Object is self-occluding!)
- For correct effects, the real normal at the point is necessary. (Normals are coming of use!)

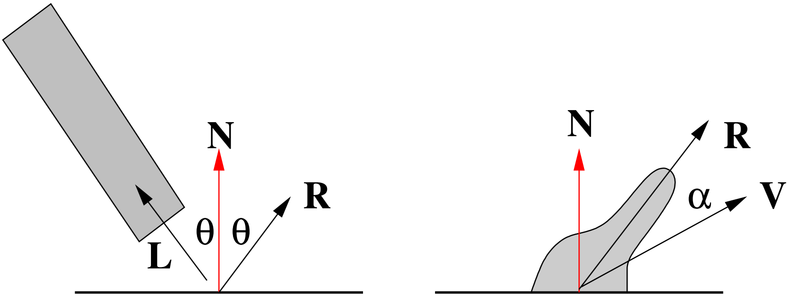
Material Colour

- What is the colour of an object?
  The light it reflects. Rest is absorbed.
- A fully red object has colour (1, 0, 0). It reflects all of the red falling on it and none of the green or blue.
- How do we represent/simulate that? ......
- We also need to know the spectral composition of the light falling on the object.
- Illumination equation: $I_{d\lambda} = I_{p\lambda}\ k_d\ O_{d\lambda}\ (\mathbf{N}\cdot\mathbf{L})$
- $k_d O_{d\lambda}$ can be called the **diffuse colour** of object.

Shiny Objects and Highlights

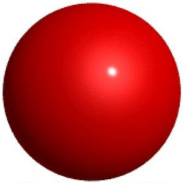
- Shiny objects (smooth metal, polished marble) behave differently.
- A ***highlight*** can be seen on them due to the light source. Highlight has the colour of the light source, irrespective of object colour.
- Highlight moves with the viewing angle. Appearance of the object point depends on the view angle.
- This is called **specular reflection**.
- A mirror is an ideal specular reflector.

Shiny Objects and Highlights (cont.)



Maximum Reflection along direction R

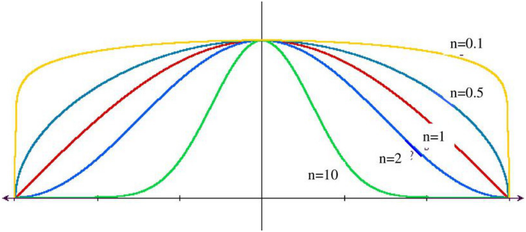
Specular Reflections



- A distinguished direction exists for reflection, depending on the incident light direction and normal direction.
- Reflection falls off quickly as view moves away from this angle.
- Reflect the light vector about the normal vector to get the specular reflection direction.

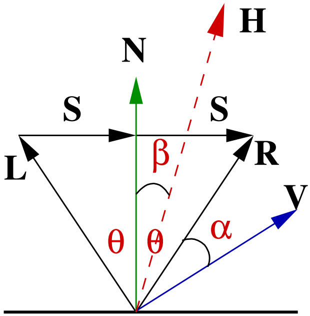
Specular Lighting: Phong Model

- α is the angle between the reflection direction R and the view direction V.
- Phong's model: $I_s = I_p\, k_s \cos^n \alpha = I_p\, k_s\, (\mathbf{V} \cdot \mathbf{R})^{\mathbf{n}}$
- As n becomes larger, reflection becomes sharper. k_s is the specular reflection coefficient.
- Sometimes, specular colour $O_{s\lambda}$ is also given to the object!



Computing Reflection Vector R

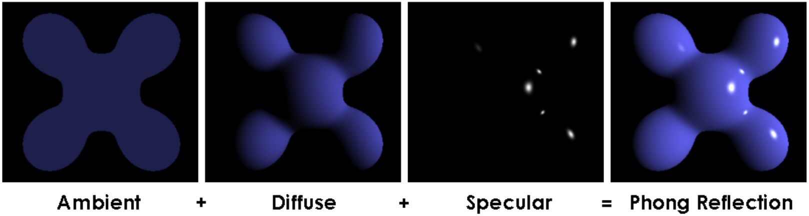
- $\mathbf{L} + \mathbf{S} = \mathbf{N}\ (\mathbf{N}{\cdot}\mathbf{L})$
- $\mathbf{R} = \ \mathbf{L} + 2\mathbf{S} = ??$
- Halfway vector **H** may also be used: $\mathbf{H} = \frac{\mathbf{L}+\mathbf{V}}{|\mathbf{L}+\mathbf{V}|}$
- Maximum highlight when **H** and **N** coincide
- Deviation from it $\mathbf{N} \cdot \mathbf{H}$ can be used instead of $\mathbf{V} \cdot \mathbf{R}$



Ambient Light

- In graphics, some light is always present.
- This is called **ambient light** I_a, present everywhere.
- The net effect of all the light that reflects from the environment etc.
- This light helps see objects even when no explicit light source is present.
- Illumination equation: $I_\lambda = I_{a\lambda}\, k_a\, O_{a\lambda} + I_{p\lambda}\, k_d\, O_{d\lambda}\, (\mathbf{N}\cdot\mathbf{L})$
- k_a is the ambient reflection coefficient.

Phong Reflection Example



Brad Smith

Atmospheric Effects

- Most straightforward is attenuation.
- The light that reaches the point $I = f_{\text{att}} I_p$.
- Physics says: $f_{\text{att}} = 1/d_L^2$ by inverse square law.
- In practice, it doesn't work well. Objects that are far become indistinguishable.
- In graphics, we use $f_{\text{att}} = 1/(c_1 + c_2\, d_L + c_2\, d_L^2)$
- Illum Equn: $I_\lambda = I_{a\lambda}\, k_a\, O_{a\lambda} + f_{\text{att}}\, I_\lambda\, k_d\, O_{d\lambda}\, (\mathbf{N} \cdot \mathbf{L})$

One Light to More

- Total illumination equation for one light source:

$$I_{p\lambda} = I_{a\lambda}\, k_a\, O_{a\lambda} + f_{\text{att}}\, I_\lambda \left[k_d\, O_{d\lambda}\, (\mathbf{N}\cdot\mathbf{L}) + k_s\, O_{s\lambda}\, (\mathbf{V}\cdot\mathbf{R})^{\mathbf{n}}\right]$$

- When multiple light sources are involved?
- Simple model: Add up the individual contributions together!
- Question: Which terms depend on the light source?

Multiple Light Sources

- Contributions of diffuse, and specular reflections are added together.
- When multiple light sources are involved:

$$I_{p\lambda} = I_{a\lambda}\, k_a\, O_{a\lambda} + \sum_i f_{\mathrm{att}_i}\, I_{\lambda i}\, [k_d\, O_{d\lambda}(\mathbf{N}\cdot\mathbf{L_i}) + k_s\, O_{s\lambda}(\mathbf{V}\cdot\mathbf{R_i})^{\mathbf{n}}]$$

Emissive Colour

- Objects that emit colour such as a tubelight as an object.
- Material property can include emissive colours for such self luminous objects.
- The emissive colour is added to every point of the object.
- Only the appearance of the object is affected.
- Emissive objects do not work as light sources automatically.

Light Sources

- When the light source is infinitely far (like the Sun), only the direction matters. *Directional Light*
- When light has a position, **L** vector can be computed from it. *Point Light Source*
- Point light sources illuminate in all directions.
- Alternately, light source can have a position, a direction, and a drop off formula. Lighting is maximum in the given direction and drops off as you move away from that direction. *Spot Light*
- Colour of the light is important to compute effects.

Light Sources in OpenGL

- Position: Set $(x, y, z, 0)$ for directional light sources, located at ∞. Finite position for others. Default: point light source.
- Each has: Ambient, Diffuse, Specular colours.
  Spot direction, cut-off, exponent. Attenuation.
- Since each polygon is drawn independently, shadows do not appear automatically.
- Read about `glLight()`, `glLightModel()`.
  And `glEnable()` for `GL_LIGHTi`, `GL_LIGHTING`.

Material in OpenGL

- colours: the material colour times the appropriate reflection coefficient.
- Ambient, Diffuse, Specular, Emissive colours.
- Shininess: like the n in the $\cos^n \alpha$ term.
- Produces good effects when combined with light sources.
- `glMaterial()` changes the current material properties. Read!

Lighting: Summary

- **Material:** Diffuse colour, specular colour, ambient colour.
- **Light Source:** Type, directionality and colour. Usually, diffuse, specular, and ambient colours could be attached to light sources.
- **Reflection:** Diffuse and Specular.
- Total illumination equation for multiple light sources:

$$I_{p\lambda} = I_{a\lambda}\, k_a\, O_{a\lambda} + \sum_i f_{\mathrm{att}_i}\, I_{\lambda i}\, [k_d\, O_{d\lambda}(\mathbf{N} \cdot \mathbf{L_i}) + k_s\, O_{s\lambda}(\mathbf{V} \cdot \mathbf{R_i})^{\mathbf{n}}]$$

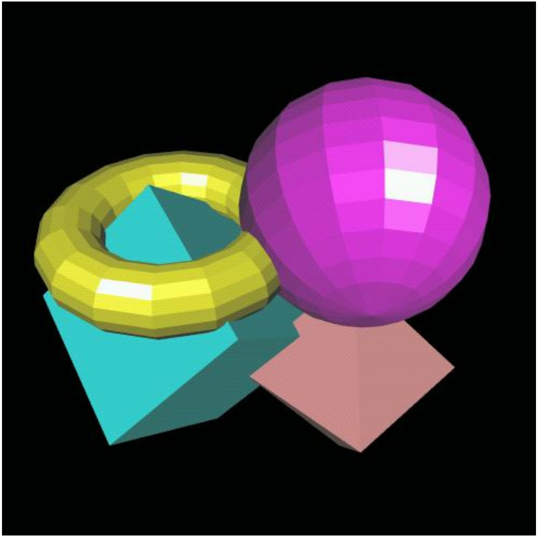
Shading Models for Polygons

- The illumination equation can be evaluated at every pixel to compute the colour/intensity there.
- This is expensive computationally.
  (Does it give the correct results?)
- Can we take advantage of the coherence or the fact that we are computing for a planar polygon?
- A number of different options exist.

Constant or Flat Shading

- Evaluate llumination equation once per polygon
- Apply the intensity values to the whole polygon
- Each polygon gets a constant colour. They look flat
- Most easy computationally
- The results will be correct if:
  $\mathbf{N} \cdot \mathbf{L}$ is constant across the polygon and
  $\mathbf{N} \cdot \mathbf{V}$ is constant across the polygon and
  Object is polyhedral, not an approximation
- Which point? Centre? First vertex? Results may vary.

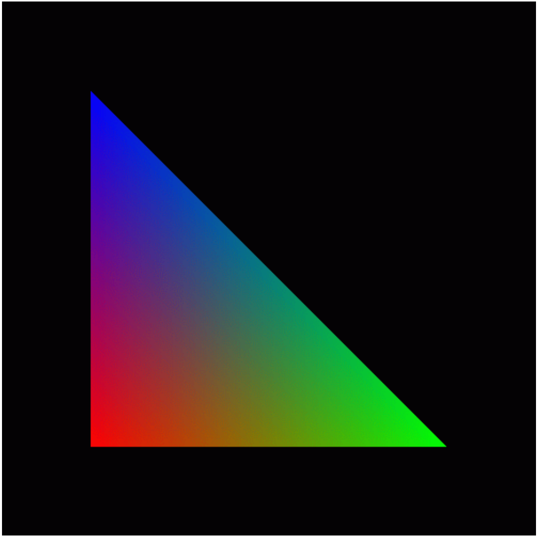
Constant or Flat Shading (cont.)



Interpolated Shading

- Results of flat shading are unsatisfactory.
- Interpolated shading assumes that properties can be calculated at the vertices and can be interpolated for the interior points.
- The interpolation can be done along with the interpolation of the Z values.
- Use the normal, light and view vectors for the vertices to compute the values.
- Since the polygon is planar, the normals should be the same!?!?

Interpolated Shading (cont.)

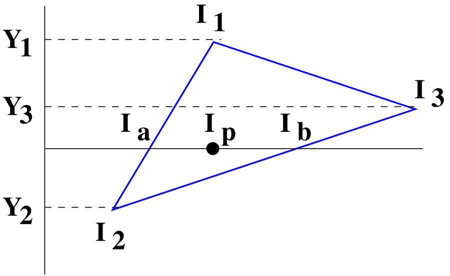


Gourard Shading

- Also called *colour interpolation shading*.
- Evaluate the illumination equation at each of the polygon vertices.
- Use exact normals – stored already or computed on the fly – if available.
- Otherwise, use the average of all polygons that meet at the vertex!
- Interpolate intensities along the edges that connect vertices.
- Interpolate along scan lines using intensities at the edges. This can be combined nicely with the computation we perform on spans of polygons nicely.

Gourard Shading (cont.)

- Produces good results, not correct according to Physics!

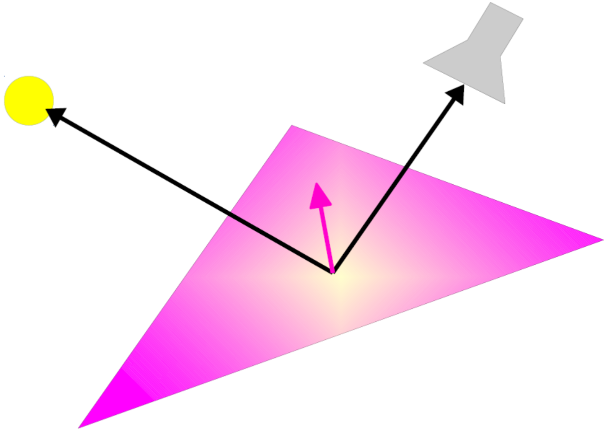


- Linear along edges for I_a, I_b and along scan line for I_p

Highlights under Interpolation

- Specular highlights are localized bright areas.
- If the highlight falls in the middle of a polygon, it will be completely missed as illumination equation is computed only for the vertices.
- If highlight falls on a vertex, it will be interpolated across, making it less local.
- These cannot be handled by interpolation of intensities.

Highlights under Interpolation



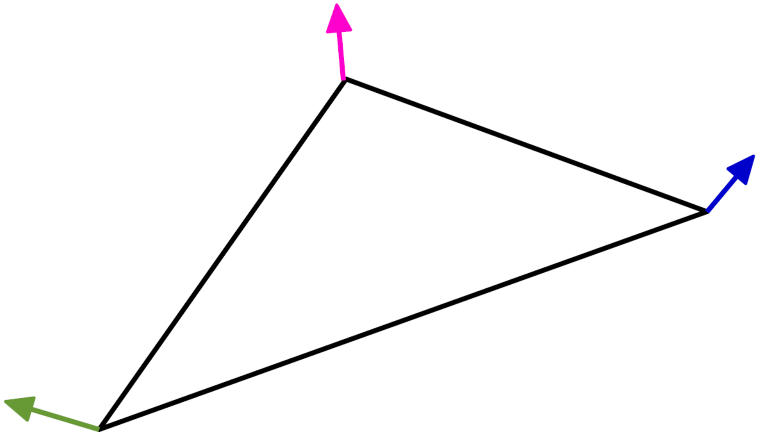
Polygon Approximation

- When a curved object is approximated using a polygon, adjacent polygons could have different intensity values.
- Flat shading will bring it out sharply; object will appear *faceted*.
- Interpolated shading may not help much if neighbouring polygons have different normals.
- Will it help to evaluate the illumination equation at each pixel?

Normals at Points

- If a parametric or analytic representation of the object that is approximated is available, exact normals can be computed at each point.
- Keep the equations along with the objects so that exact normals can be computed.
- Alternately, compute the normals for each vertex when converting the smooth object to a polygon mesh.
- Keeping the normal vector at each vertex along with the 3D coordinates is quite popular.

Normals at Points (cont.)

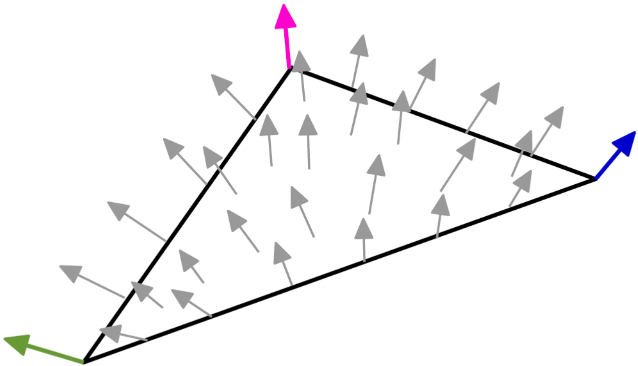


- **Normals could be from the underlying exact surface.**

Phong Shading

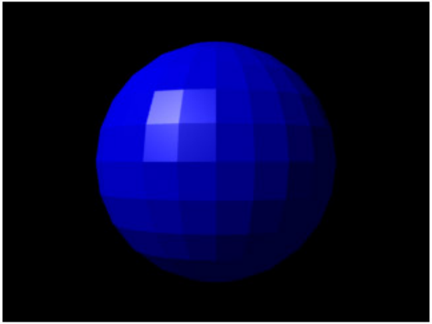
- Interpolate surface normals across the polygon, given the normals at the vertices.
- Compute illumination equation with these interpolated normals for each pixel.
- Computes highlights very well.
- Computation time is more as the equation is evaluated at every pixel.
- Interpolation of normals is not physically based!
- Also called *normal vector interpolation shading*

Phong Shading (cont.)

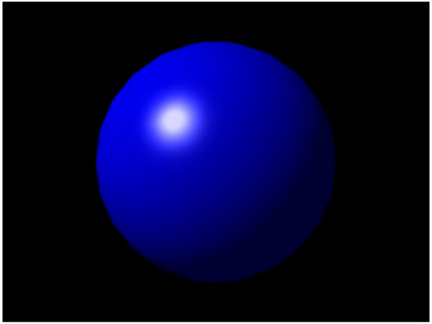


- Exact normals (not interpolated) can be evaluated at each pixel in the fragment shader for exact lighting.

Phong Shading

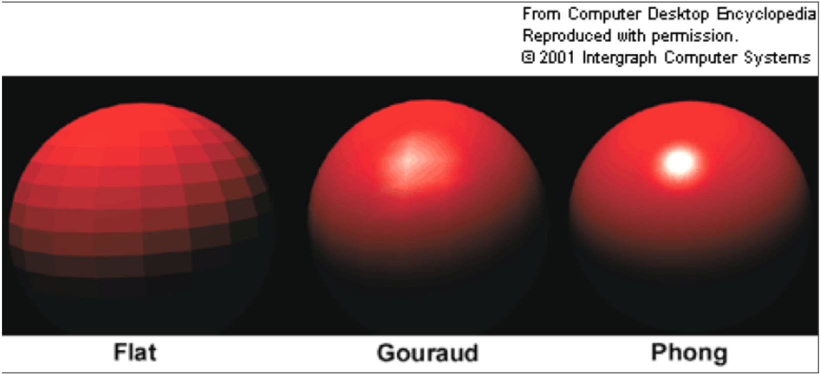


FLAT SHADING



PHONG SHADING

Flat vs Goroud vs Phong Shading



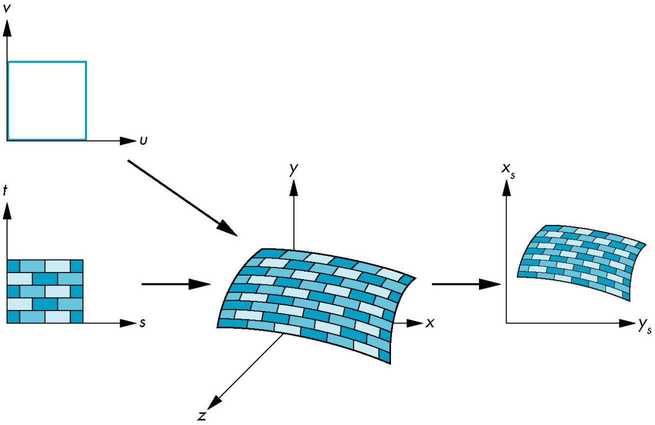
Texture Mapping

- Shading can produce only uniform or smooth surfaces. They look plastic and artificial.
- Map a real or synthetic image onto a polygon surface.
- Each 3D point is also associated with a 2D texture coordinate pair.
- These refer to points in the image to associate with the points.
- While scan converting the polygon, the pixel coordinates are mapped to the texture image coordinates.
- The texture colour for the pixel is obtained by interpolating the texture image using these coordinates.
- Finally, a texture colour is obtained for the pixel.

Texture Mapping (cont.)

M. Bronstein:ECCV-2016 Tutorial

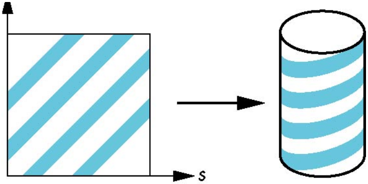
Texture Mapping



Source: Interactive Computer Graphics 4E Addison-Wesley 2005

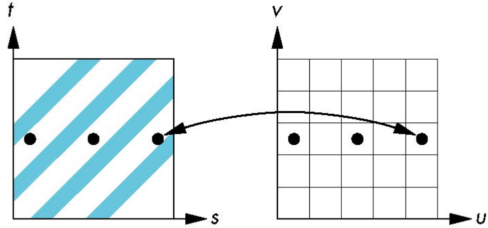
Texture Mapping

- *Backward Mapping:* Given a 3D point, find mapping to a 2D coordinate in texture image. Generally hard to find.
- *Two-part Mapping:* First map texture to a regular 3D model (e.g., Sphere or Cylinder and then eventually map from 3D model to actual object.



Source: Interactive Computer Graphics 4E Addison-Wesley 2005

Texture Mapping: Aliasing



Source: Interactive Computer Graphics 4E Addison-Wesley 2005

Transparency

- How do we deal with transparent objects?
- Need to model Refraction.
- Ignore them! Leads to non-refractive transparency, which is still OK.
- Refraction takes considerable effort to handle. We will see it later.
- Other tricks: Interpolated transparency

Interpolated Transparency

- When two objects overlap at a pixel, use a weighted average of the two colours at the pixel.
- $I_\lambda = (1 - k_{t1})\, I_{\lambda 1} + k_{t1}\, I_{\lambda 2}$
- k gives the transmission coefficient or the transparency of each polygon. If $k_{t1} = 0$, polygon 1 is totally opaque.
- $(1 - k)$ is opacity.

RGBA Colours

- Graphics systems (OpenGL, for example) use 4-component colours, with A being the α channel.
- Alpha measures the opacity of the colour. Equals 1 for completely opaque objects and 0 for totally transparent ones.
- A similar interpolation formula is used when an object with alpha is drawn on top of another object.
- If the Z-buffer test succeeds, the colours of the new polygon are blended into the frame buffer using α instead of replacing it entirely.

Shadows

- Shadows are the results of light **not** reaching some part of the scene.
- Sufaces that are not visible from the light sources are in shadow. VSD algorithms can be used to determine this.
- **Shadow Map:** A bitmap in image space containing projections of shadowed regions.
- Modify the illumination equation to:

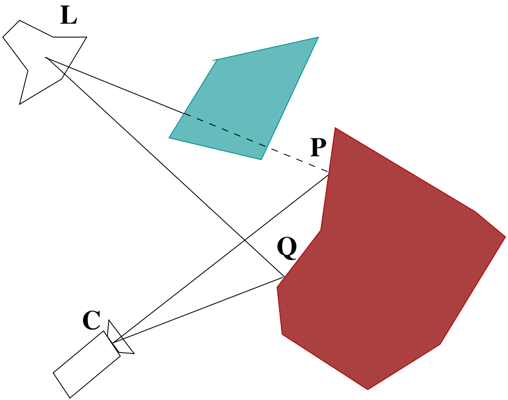
$$I_{p\lambda} = I_{a\lambda} k_a O_{a\lambda} + \sum_{1 \le i \le m} S_i f_{\mathrm{att}_i} I_{\lambda i} \{ k_d O_{d\lambda} (\mathbf{N} \cdot \mathbf{L_i}) + k_s O_{s\lambda} (\mathbf{V} \cdot \mathbf{R_i})^{\mathbf{n}} \}$$

- Shadow map S_i indicates if the pixel is under the shadow for the light source i. $S_i = 0$ if under shadow, $S_i = 1$ otherwise.

2-Pass Z-buffer Shadow Algorithm

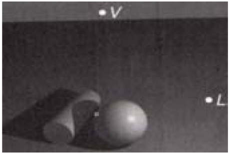
- Draw the scene with light source as the camera using Z-buffering.
- Read the depth values from the Z-buffer into a z_l buffer. This is the *Shadow Map*.
- Draw scene from camera's viewpoint with Z-buffering.
- Transform each visible point (x_o, y_o, z_o) to (x_o', y_o', z_o') in the light source's coordinates.
- If $z_o' > z_l(x_o', y_o')$, there is another point that is closer to the light source. Do not light the pixel.

2-Pass Z-buffer Shadow Algorithm (cont.)

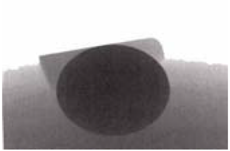


- Point P will be shadowed and Q will not be.

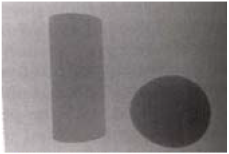
2-Pass Z-buffer Shadow Algorithm (cont.)



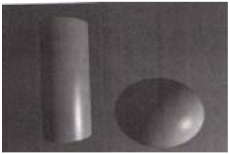
Overview



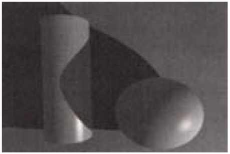
Light's Z-buffer



Observer's Z-buffer



Observer's Image



With shadows

Source: Foley & Van Dam