University of British Columbia
CPSC 314 Computer Graphics
Jan-Apr 2013

Tamara Munzner

Lighting/Shading

http://www.ugrad.cs.ubc.ca/~cs314/Vjan2013

Correction: W2V vs. V2W

slide 26, Viewing

- $M_{V2W}=(M_{W2V})^{-1}=R^{-1}T^{-1}$

$$M_{view2world} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} u_x & u_y & u_z & -\mathbf{e}\bullet\mathbf{u} \\ v_x & v_y & v_z & -\mathbf{e}\bullet\mathbf{v} \\ w_x & w_y & w_z & -\mathbf{e}\bullet\mathbf{w} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_{V2W} = \begin{bmatrix} u_x & u_y & u_z & -e_x*u_x + -e_y*u_y + -e_z*u_z \\ v_x & v_y & v_z & -e_x*v_x + -e_y*v_y + -e_z*v_z \\ w_x & w_y & w_z & -e_x*w_x + -e_y*w_y + -e_z*w_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2

Recorrection: Perspective Derivation

slide 91, Viewing

L/R sign error

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} E & 0 & A & 0 \\ 0 & F & B & 0 \\ 0 & 0 & C & D \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$x' = Ex + Az$
$y' = Fy + Bz$
$z' = Cz + D$
$w' = -z$

$x = left \rightarrow x'/w' = -1$
$x = right \rightarrow x'/w' = 1$
$y = top \rightarrow y'/w' = 1$
$y = bottom \rightarrow y'/w' = -1$
$z = -near \rightarrow z'/w' = -1$
$z = -far \rightarrow z'/w' = 1$

z axis flip!

$$y' = Fy + Bz, \quad \frac{y'}{w'} = \frac{Fy+Bz}{w'}, \quad 1 = \frac{Fy+Bz}{w'}, \quad 1 = \frac{Fy+Bz}{-z},$$

$$1 = F\frac{y}{-z} + B\frac{z}{-z}, \quad 1 = F\frac{y}{-z} - B, \quad 1 = F\frac{top}{-(-near)} - B,$$

$$1 = F\frac{top}{near} - B$$

3

Reading for This Module

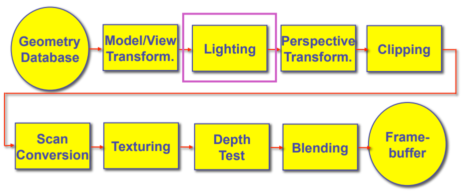
- FCG Chapter 10 Surface Shading
- FCG Section 8.2.4-8.2.5
- RB Chap Lighting

4

Lighting I

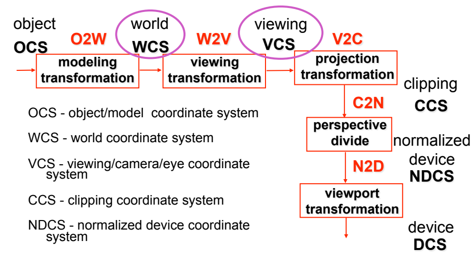
5

Rendering Pipeline



6

Projective Rendering Pipeline



OCS - object/model coordinate system
WCS - world coordinate system
VCS - viewing/camera/eye coordinate system
CCS - clipping coordinate system
NDCS - normalized device coordinate system
DCS - device/display/screen coordinate system

7

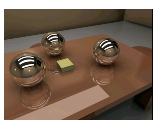
Goal

- simulate interaction of light and objects
- fast: fake it!
  - approximate the look, ignore real physics
- get the physics (more) right
  - BRDFs: Bidirectional Reflection Distribution Functions
- local model: interaction of each object with light
- global model: interaction of objects with each other

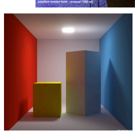

8

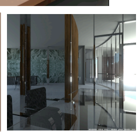
Photorealistic Illumination

- transport of energy from light sources to surfaces & points
  - global includes direct and indirect illumination – more later

[electricimage.com]





Henrik Wann Jensen

9

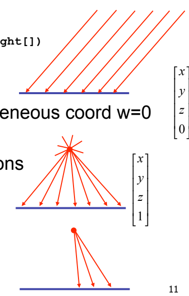
Illumination in the Pipeline

- local illumination
  - only models light arriving directly from light source
  - no interreflections or shadows
    - can be added through tricks, multiple rendering passes
- light sources
  - simple shapes
- materials
  - simple, non-physical reflection models

10

Light Sources

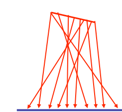
- types of light sources
  - `glLightfv(GL_LIGHT0,GL_POSITION,light[])`
- directional/parallel lights
  - real-life example: sun
  - infinitely far source: homogeneous coord w=0
- point lights
  - same intensity in all directions
- spot lights
  - limited set of directions:
    - point+direction+cutoff angle



11

Light Sources

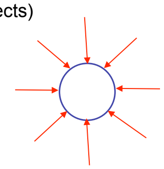
- area lights
- light sources with a finite area
- more realistic model of many light sources
- not available with projective rendering pipeline (i.e., not available with OpenGL)



12

Light Sources

- ambient lights
- no identifiable source or direction
- hack for replacing true global illumination
  - (diffuse interreflection: light bouncing off from other objects)



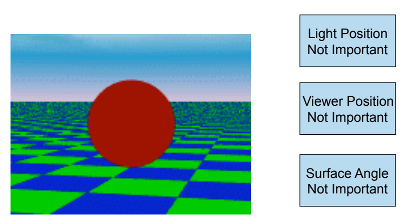
13

Diffuse Interreflection

14

Ambient Light Sources

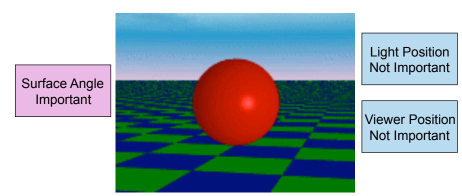
- scene lit only with an ambient light source



15

Directional Light Sources

- scene lit with directional and ambient light



16

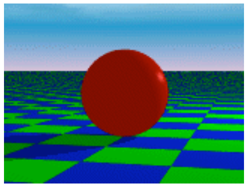
Point Light Sources

- scene lit with ambient and point light source

Light Position Important

Viewer Position Important

Surface Angle Important

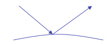


Light Sources

- geometry: positions and directions
  - standard: world coordinate system
    - effect: lights fixed wrt world geometry
    - demo: http://www.xmission.com/~nate/tutors.html
  - alternative: camera coordinate system
    - effect: lights attached to camera (car headlights)
  - points and directions undergo normal model/ view transformation
- illumination calculations: camera coords

Types of Reflection

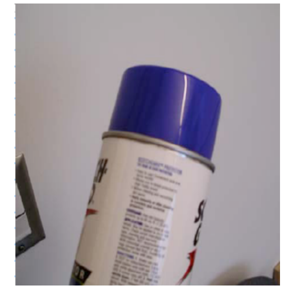
- *specular* (a.k.a. *mirror* or *regular*) reflection causes light to propagate without scattering.



- *diffuse* reflection sends light in all directions with equal energy.

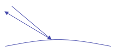
- *mixed* reflection is a weighted combination of specular and diffuse.

Specular Highlights

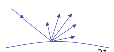


Types of Reflection

- *retro-reflection* occurs when incident energy reflects in directions close to the incident direction, for a wide range of incident directions.

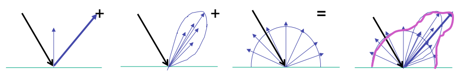


- *gloss* is the property of a material surface that involves mixed reflection and is responsible for the mirror like appearance of rough surfaces.



Reflectance Distribution Model

- most surfaces exhibit complex reflectances
  - vary with incident and reflected directions.
  - model with combination

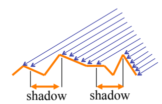


specular + glossy + diffuse = reflectance distribution

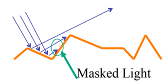
Surface Roughness

- at a microscopic scale, all real surfaces are rough

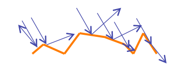
- cast shadows on themselves



- "mask" reflected light:



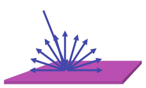
Surface Roughness

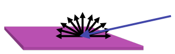


- notice another effect of roughness:
  - each "microfacet" is treated as a perfect mirror.
  - incident light reflected in different directions by different facets.
  - end result is mixed reflectance.
    - smoother surfaces are more specular or glossy.
    - random distribution of facet normals results in diffuse reflectance.

Physics of Diffuse Reflection

- ideal diffuse reflection
  - very rough surface at the microscopic level
    - real-world example: chalk
  - microscopic variations mean incoming ray of light equally likely to be reflected in any direction over the hemisphere
  - what does the reflected intensity depend on?



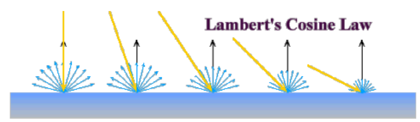


Lambert's Cosine Law

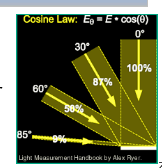
- ideal diffuse surface reflection

  the energy reflected by a small portion of a surface from a light source in a given direction is proportional to the cosine of the angle between that direction and the surface normal
- reflected intensity
  - independent of viewing direction
  - depends on surface orientation wrt light
- often called Lambertian surfaces

Lambert's Law

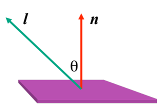


intuitively: cross-sectional area of the "beam" intersecting an element of surface area is smaller for greater angles with the normal.



Computing Diffuse Reflection

- depends on angle of incidence: angle between surface normal and incoming light
  - $I_{diffuse} = k_d \, I_{light} \cos\theta$
- in practice use vector arithmetic
  - $I_{diffuse} = k_d \, I_{light} \, (\mathbf{n} \cdot \mathbf{l})$



- **always normalize vectors used in lighting!!!**
  - **n, l** should be unit vectors
- scalar (B/W intensity) or 3-tuple or 4-tuple (color)
  - k_d: diffuse coefficient, surface color
  - I_{light}: incoming light intensity
  - $I_{diffuse}$: outgoing light intensity (for diffuse reflection)

Diffuse Lighting Examples

- Lambertian sphere from several lighting angles:

- need only consider angles from 0° to 90°
  - *why?*
  - *demo: Brown exploratory on reflection*
  - http://www.cs.brown.edu/exploratories/freeSoftware/repository/edu/brown/cs/exploratories/applets/reflection2D/reflection_2d_java_browser.html

Specular Highlights

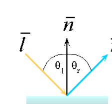
Michiel van de Panne

Physics of Specular Reflection

- at the microscopic level a specular reflecting surface is very smooth
- thus rays of light are likely to bounce off the microgeometry in a mirror-like fashion
- the smoother the surface, the closer it becomes to a perfect mirror

Optics of Reflection

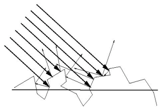
- reflection follows *Snell's Law:*
  - incoming ray and reflected ray lie in a plane with the surface normal
  - angle the reflected ray forms with surface normal equals angle formed by incoming ray and surface normal



$\theta_{(l)ight} = \theta_{(r)eflection}$

Non-Ideal Specular Reflectance

- Snell's law applies to perfect mirror-like surfaces, but aside from mirrors (and chrome) few surfaces exhibit perfect specularity
- how can we capture the "softer" reflections of surface that are glossy, not mirror-like?
- one option: model the microgeometry of the surface and explicitly bounce rays off of it
- or…

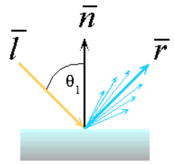


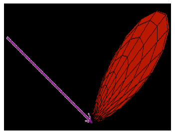
Empirical Approximation

- we expect most reflected light to travel in direction predicted by Snell's Law
- but because of microscopic surface variations, some light may be reflected in a direction slightly off the ideal reflected ray
- as angle from ideal reflected ray increases, we expect less light to be reflected

Empirical Approximation

- angular falloff





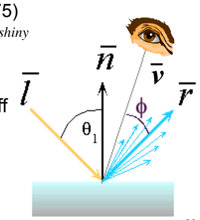
- how might we model this falloff?

Phong Lighting

- most common lighting model in computer graphics
  - (Phong Bui-Tuong, 1975)

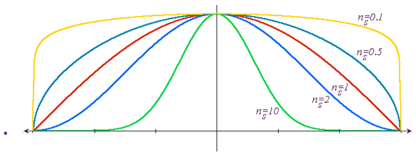
$$I_{specular} = k_s I_{light} (\cos\phi)^{n_{shiny}}$$

- n_{shiny} : purely empirical constant, varies rate of falloff
- k_s: specular coefficient, highlight color
- no physical basis, works ok in practice



Phong Lighting: The n_{shiny} Term

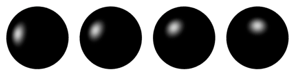
- Phong reflectance term drops off with divergence of viewing angle from ideal reflected ray



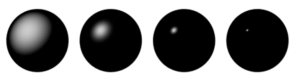
Viewing angle – reflected angle

Phong Examples

varying l



varying n_{shiny}

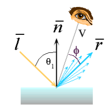


Calculating Phong Lighting

- compute **cosine** term of Phong lighting with vectors

$$I_{specular} = k_s I_{light} (v \bullet r)^{n_{shiny}}$$

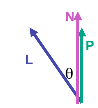
- v: unit vector towards viewer/eye
- r: ideal reflectance direction (unit vector)
- k_s: specular component
  - highlight color
- I_{light}: incoming light intensity



- how to efficiently calculate **r** ?

Calculating R Vector

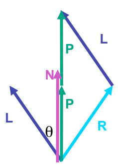
P = N cos θ |**L**| |**N**| projection of **L** onto **N**
P = N cos θ **L, N** are unit length
P = N (N · L)



Calculating R Vector

P = N cos θ |**L**| |**N**| projection of **L** onto **N**
P = N cos θ **L, N** are unit length
P = N (N · L)

2 P = R + L
2 P – L = R
2 (N (N · L)) - L = R



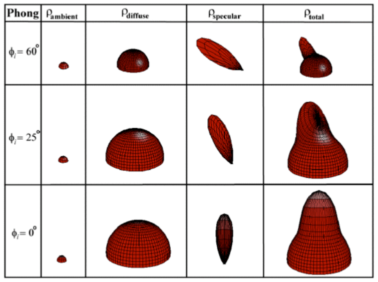
Phong Lighting Model

- combine ambient, diffuse, specular components

$$I_{total} = k_a I_{ambient} + \sum_{i=1}^{\#lights} I_i \left(k_d (n \bullet l_i) + k_s (v \bullet r_i)^{n_{shiny}}\right)$$

- commonly called *Phong lighting*
  - once per light
  - once per color component
- reminder: normalize your vectors when calculating!
  - normalize all vectors: n,l,r,v

Phong Lighting: Intensity Plots

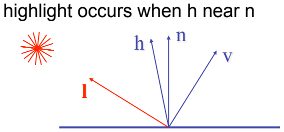


Blinn-Phong Model

- variation with better physical interpretation
  - Jim Blinn, 1977

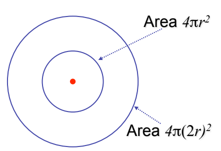
$$I_{out}(\mathbf{x}) = I_{in}(\mathbf{x})(\mathbf{k}_s(\mathbf{h} \bullet \mathbf{n})^{n_{shiny}});\text{with } \mathbf{h} = (\mathbf{l} + \mathbf{v})/2$$

- ***h***: halfway vector
  - h must also be explicitly normalized: h / |h|
  - highlight occurs when h near n



Light Source Falloff

- quadratic falloff
- brightness of objects depends on power per unit area that hits the object
- the power per unit area for a point or spot light decreases quadratically with distance



Light Source Falloff

- non-quadratic falloff
- many systems allow for other falloffs
- allows for faking effect of area light sources
- OpenGL / graphics hardware
  - I_o: intensity of light source
  - $\boldsymbol{x}$: object point
  - r: distance of light from $\boldsymbol{x}$

$$I_{in}(\mathbf{x}) = \frac{1}{ar^2 + br + c} \cdot I_0$$

Lighting Review

- lighting models
- ambient
  - normals don't matter
- Lambert/diffuse
  - angle between surface normal and light
- Phong/specular
  - surface normal, light, and viewpoint

Lighting in OpenGL

- light source: amount of RGB light emitted
  - value represents percentage of full intensity e.g., (1.0,0.5,0.5)
  - every light source emits ambient, diffuse, and specular light
- materials: amount of RGB light reflected
  - value represents percentage reflected e.g., (0.0,1.0,0.5)
- interaction: multiply components
  - red light (1,0,0) x green surface (0,1,0) = black (0,0,0)

Lighting in OpenGL

```
glLightfv(GL_LIGHT0, GL_AMBIENT, amb_light_rgba );
glLightfv(GL_LIGHT0, GL_DIFFUSE, dif_light_rgba );
glLightfv(GL_LIGHT0, GL_SPECULAR, spec_light_rgba );
glLightfv(GL_LIGHT0, GL_POSITION, position);
glEnable(GL_LIGHT0);

glMaterialfv( GL_FRONT, GL_AMBIENT, ambient_rgba );
glMaterialfv( GL_FRONT, GL_DIFFUSE, diffuse_rgba );
glMaterialfv( GL_FRONT, GL_SPECULAR, specular_rgba );
glMaterialfv( GL_FRONT, GL_SHININESS, n );
```

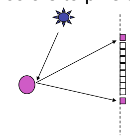
- warning: glMaterial is expensive and tricky
  - use cheap and simple glColor when possible
  - see OpenGL Pitfall #14 from Kilgard's list

http://www.opengl.org/resources/features/KilgardTechniques/oglpitfall/

Shading

Lighting vs. Shading

- **lighting**
  - process of computing the luminous intensity (i.e., outgoing light) at a particular 3-D point, usually on a surface
- **shading**
  - the process of assigning colors to pixels
  - (why the distinction?)



Applying Illumination

- we now have an illumination model for a point on a surface
- if surface defined as mesh of polygonal facets, *which points should we use?*
  - fairly expensive calculation
  - several possible answers, each with different implications for visual quality of result

Applying Illumination

- polygonal/triangular models
  - each facet has a constant surface normal
  - if light is directional, diffuse reflectance is constant across the facet
  - why?

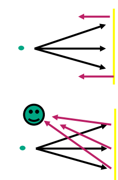
Flat Shading

- simplest approach calculates illumination at a single point for each polygon

- obviously inaccurate for smooth surfaces

Flat Shading Approximations

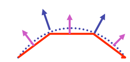
- if an object really <u>is</u> faceted, is this accurate?
- no!
  - for point sources, the direction to light varies across the facet
  - for specular reflectance, direction to eye varies across the facet



Improving Flat Shading

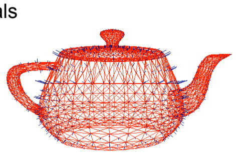
- what if evaluate Phong lighting model at each pixel of the polygon?
  - better, but result still clearly faceted

- for smoother-looking surfaces we introduce *vertex normals* at each vertex
  - usually different from facet normal
  - used *only* for shading
  - think of as a better approximation of the *real* surface that the polygons approximate



Vertex Normals

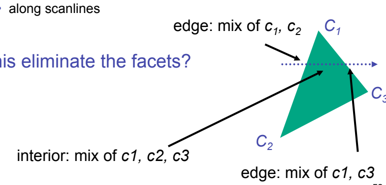
- vertex normals may be
  - provided with the model
  - computed from first principles
  - approximated by averaging the normals of the facets that share the vertex



Gouraud Shading

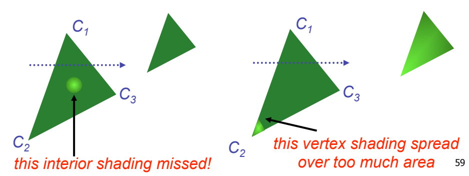
- most common approach, and what OpenGL does
  - perform Phong lighting at the vertices
  - linearly interpolate the resulting colors over faces
    - along edges
    - along scanlines

does this eliminate the facets?



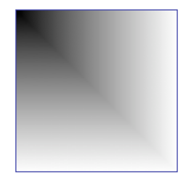
Gouraud Shading Artifacts

- often appears dull, chalky
- lacks accurate specular component
  - if included, will be averaged over entire polygon



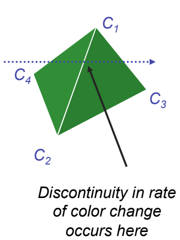
Gouraud Shading Artifacts

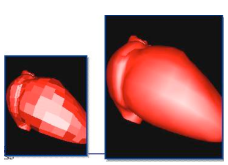
- Mach bands
  - eye enhances discontinuity in first derivative
  - very disturbing, especially for highlights



Gouraud Shading Artifacts

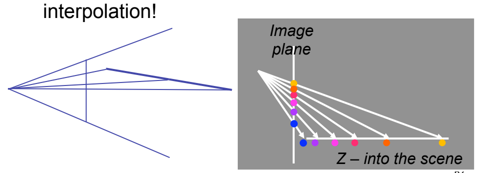
- Mach bands





Gouraud Shading Artifacts

- perspective transformations
  - affine combinations only invariant under affine, **not** under perspective transformations
  - thus, perspective projection alters the linear interpolation!



Gouraud Shading Artifacts

- perspective transformation problem
  - colors slightly "swim" on the surface as objects move relative to the camera
  - usually ignored since often only small difference
    - usually smaller than changes from lighting variations
  - to do it right
    - either shading in object space
    - or correction for perspective foreshortening
    - expensive – thus hardly ever done for colors

Phong Shading

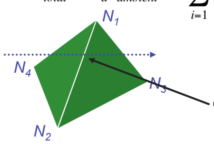
- linearly interpolating surface normal across the facet, applying Phong lighting model at every pixel
  - same input as Gouraud shading
  - pro: much smoother results
  - con: considerably more expensive

- **not** the same as Phong lighting
  - common confusion
  - Phong lighting: empirical model to calculate illum a point on a surface

Phong Shading

- linearly interpolate the vertex normals
  - compute lighting equations at each pixel
  - can use specular component

$$I_{total} = k_a I_{ambient} + \sum_{i=1}^{\#lights} I_i \left(k_d (\mathbf{n}\cdot\mathbf{l_i}) + k_s (\mathbf{v}\cdot\mathbf{r_i})^{n_{shiny}} \right)$$



remember: normals used in diffuse and specular terms

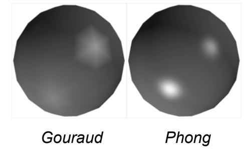
discontinuity in normal's rate of change harder to detect

Phong Shading Difficulties

- computationally expensive
  - per-pixel vector normalization and lighting computation!
  - floating point operations required
- lighting after perspective projection
  - messes up the angles between vectors
  - have to keep eye-space vectors around
- no direct support in pipeline hardware
  - but can be simulated with texture mapping
  - stay tuned for modern hardware: shaders

Shading Artifacts: Silhouettes

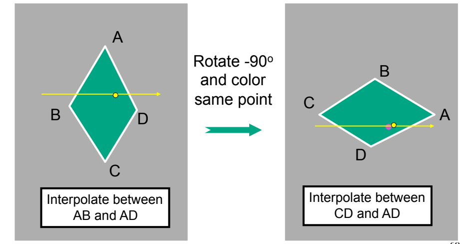
- polygonal silhouettes remain



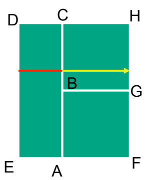
Gouraud *Phong*

Shading Artifacts: Orientation

- interpolation dependent on polygon orientation
  - view dependence!



Shading Artifacts: Shared Vertices



vertex B shared by two rectangles on the right, but not by the one on the left

first portion of the scanline is interpolated between DE and AC

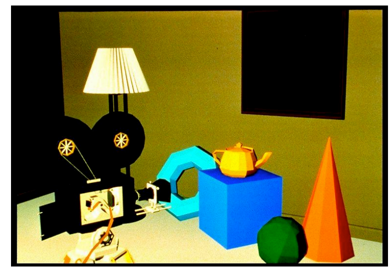
second portion of the scanline is interpolated between BC and GH

a large discontinuity could arise

Shading Models Summary

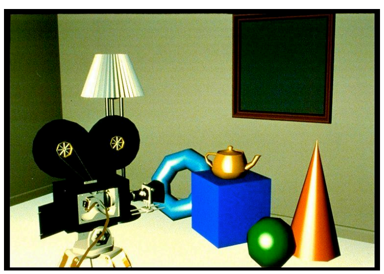
- flat shading
  - compute Phong lighting once for entire polygon
- Gouraud shading
  - compute Phong lighting at the vertices and interpolate lighting values across polygon
- Phong shading
  - compute averaged vertex normals
  - interpolate normals across polygon and perform Phong lighting across polygon

Shutterbug: Flat Shading



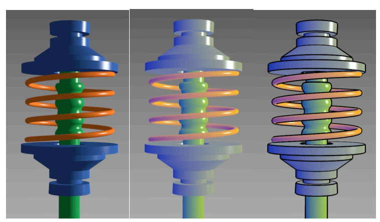
Shutterbug: Gouraud Shading

Shutterbug: Phong Shading



Non-Photorealistic Shading

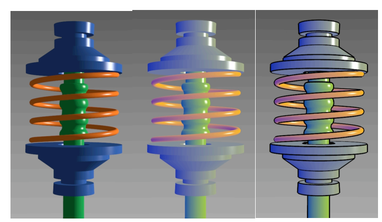
- cool-to-warm shading $k_w = \frac{1+\mathbf{n}\cdot\mathbf{l}}{2}, c = k_w c_w + (1-k_w)c_c$



http://www.cs.utah.edu/~gooch/SIG98/paper/drawing.html

Non-Photorealistic Shading

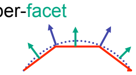
- draw silhouettes: if $(\mathbf{e}\cdot\mathbf{n_0})(\mathbf{e}\cdot\mathbf{n_1}) \le 0$, **e**=edge-eye vector
- draw creases: if $(\mathbf{n_0}\cdot\mathbf{n_1}) \le threshold$

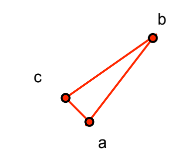


http://www.cs.utah.edu/~gooch/SIG98/paper/drawing.html

Computing Normals

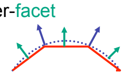
- per-vertex normals by interpolating per-facet normals
  - OpenGL supports both
- computing normal for a polygon

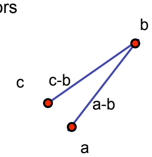




Computing Normals

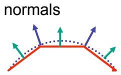
- per-vertex normals by interpolating per-facet normals
  - OpenGL supports both
- computing normal for a polygon
  - three points form two vectors

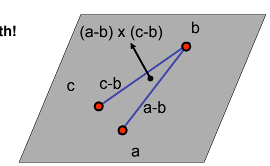




Computing Normals

- per-vertex normals by interpolating per-facet normals
  - OpenGL supports both
- computing normal for a polygon
  - three points form two vectors
  - cross: normal of plane gives direction
  - **normalize to unit length!**
  - which side is up?
    - convention: points in counterclockwise order





Specifying Normals

- OpenGL state machine
  - uses last normal specified
  - if no normals specified, assumes all identical
- per-vertex normals

```
glNormal3f(1,1,1);
glVertex3f(3,4,5);
glNormal3f(1,1,0);
glVertex3f(10,5,2);
```

- per-face normals

```
glNormal3f(1,1,1);
glVertex3f(3,4,5);
glVertex3f(10,5,2);
```