University of British Columbia
CPSC 314 Computer Graphics
May-June 2005

Tamara Munzner

Lighting/Shading I, II, III

Week 3, Tue May 24

http://www.ugrad.cs.ubc.ca/~cs314/Vmay2005

News

- P1 demos if you missed them
  - 3:30-4:30 today

2

Homework 2 Clarification

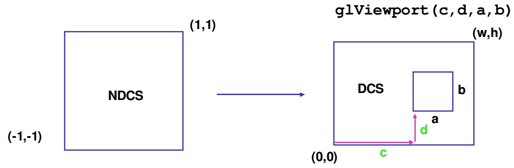
- off-by-one problem in Q4-6
  - Q4 should refer to result of Q1
  - Q5 should refer to result of Q2
  - Q6 should refer to result of Q3
- acronym confusion
  - Q1 uses W2C, whereas notes say W2V
    - world to camera/view/eye
  - Q2 uses C2P, whereas notes say V2C, C2N
  - Q3 uses N2V, whereas notes say N2D
    - normalized device to viewport/device

3

Clarification: N2D General Formulation

- translate, scale, reflect

```
glViewport(c,d,a,b);
```



- $x_D = (a*x_N)/2 + (a/2)+c$
- $y_D = - ((b*y_N)/2 + (b/2)+d)$
- $z_D = z_N/2 + 1$

4

Reading: Today

- FCG Chap 8, Surface Shading, p 141-150
- RB Chap Lighting

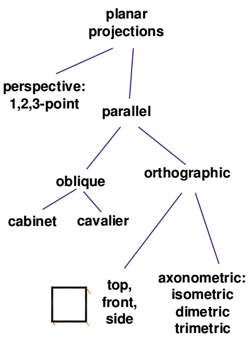
5

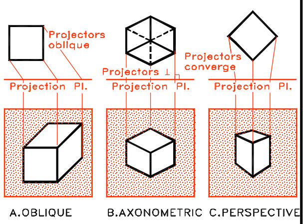
Reading: Next Time

- FCG Chap 11.1-11.4
- FCG Chap 13
- RB Chap Blending, Antialiasing, Fog, Polygon Offsets
  - only Section Blending

6

Review: Projection Taxonomy

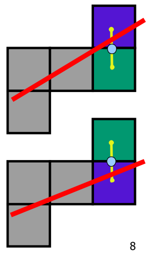




http://ceprofs.tamu.edu/tkramer/ENGR%20111/5.1/20

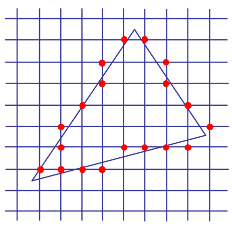
Review: Midpoint Algorithm

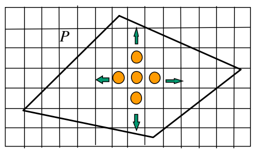
- moving horizontally along x direction
  - draw at current y value, or move up vertically to y+1?
    - check if midpoint between two possible pixel centers above or below line
- candidates
  - top pixel: (x+1,y+1)
  - bottom pixel: (x+1, y)
- midpoint: (x+1, y+.5)
- check if midpoint above or below line
  - below: top pixel
  - above: bottom pixel
- key idea behind Bresenham
  - [demo]



Review: Flood Fill

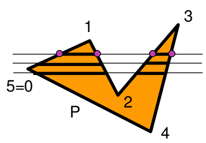
- simple algorithm
  - draw edges of polygon
  - use flood-fill to draw interior

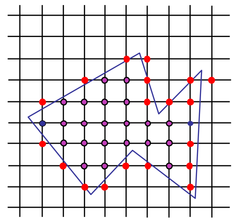




Review: Scanline Algorithms

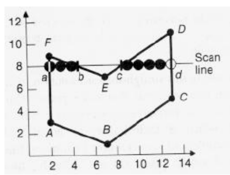
- scanline: a line of pixels in an image
  - set pixels inside polygon boundary along horizontal lines one pixel apart vertically





Review: General Polygon Rasterization

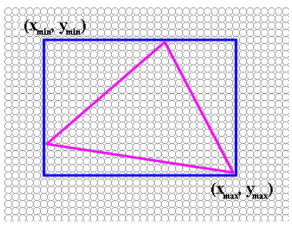
- idea: use a parity test




```
for each scanline
    edgeCnt = 0;
    for each pixel on scanline (l to r)
        if (oldpixel->newpixel crosses edge)
            edgeCnt ++;
        // draw the pixel if edgeCnt odd
        if (edgeCnt % 2)
            setPixel(pixel);
```

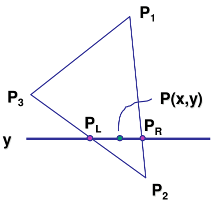

Review: Making It Fast: Bounding Box

- smaller set of candidate pixels
  - loop over xmin, xmax and ymin,ymax instead of all x, all y



Review: Bilinear Interpolation

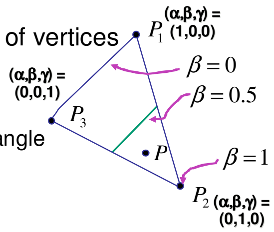
- interpolate quantity along L and R edges, as a function of y
  - then interpolate quantity as a function of x



13

Review: Barycentric Coordinates

- weighted combination of vertices
  - smooth mixing
  - speedup
    - compute once per triangle



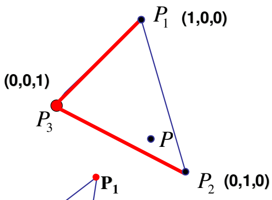
$$P = \alpha \cdot P_1 + \beta \cdot P_2 + \gamma \cdot P_3$$
$$\alpha + \beta + \gamma = 1$$
$$0 \le \alpha, \beta, \gamma \le 1$$ for points inside triangle

"convex combination of points"

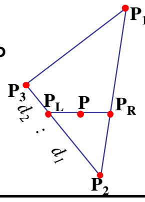
14

Review: Deriving Barycentric Coordinates

- non-orthogonal coordinate system
  - P_3 is origin, P_2-P_3, P_1-P_3 are basis vectors



- from bilinear interpolation of point P on scanline

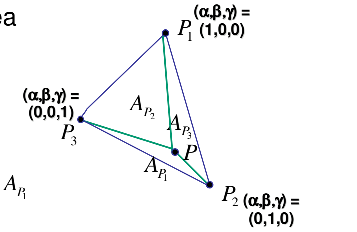


15

Correction/Review: Deriving Barycentric Coordinates

- 2D triangle area

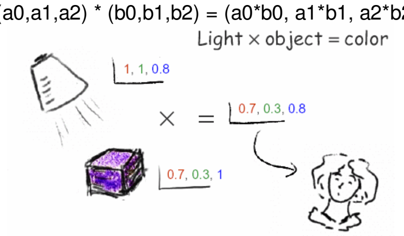
$$\alpha = A_{P1} / A$$
$$\beta = A_{P_2} / A$$
$$\gamma = A_{P3} / A$$
$$A = +A_{P_3} + A_{P_2} + A_{P_1}$$



16

Review: Simple Model of Color

- simple model based on RGB triples
- component-wise multiplication of colors
  - (a0,a1,a2) * (b0,b1,b2) = (a0*b0, a1*b1, a2*b2)

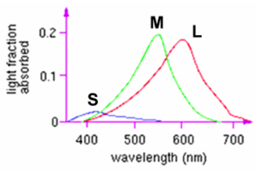


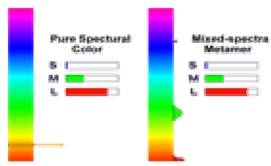
  - why does this work?

17

Review: Trichromacy and Metamers

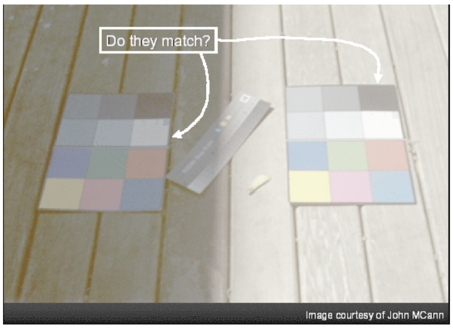
- three types of cones
- color is combination of cone stimuli
  - metamer: identically perceived color caused by very different spectra





18

Review: Color Constancy



19

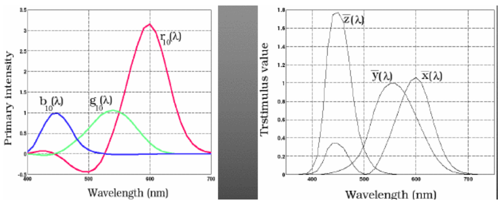
Clarification/Review: Stroop Effect

- blue
- green
- purple
- red
- orange

- say what color the text is as fast as possible
- interplay between cognition and perception

20

Review: Measured vs. CIE Color Spaces

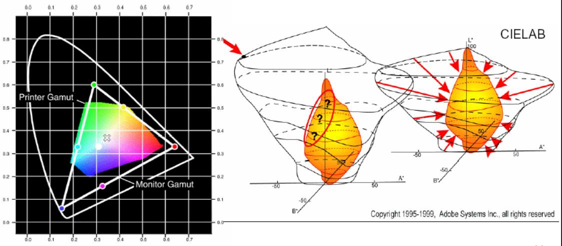


- measured basis
  - monochromatic lights
  - physical observations
  - negative lobes
- transformed basis
  - “imaginary” lights
  - all positive, unit area
  - Y is luminance

21

Review: Device Color Gamuts

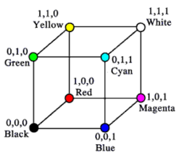
- compare gamuts on CIE chromaticity diagram
  - gamut mapping



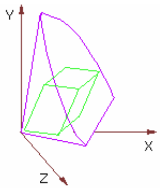
22

Review: RGB Color Space

- define colors with (r, g, b) amounts of red, green, and blue
  - used by OpenGL



- RGB color cube sits within CIE color space
  - subset of perceivable colors

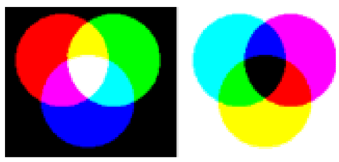


23

Review: Additive vs. Subtractive Colors

- additive: light
  - monitors, LCDs
  - RGB model
- subtractive: pigment
  - printers
  - CMY model

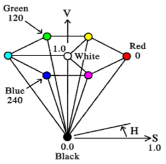
$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

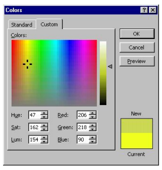


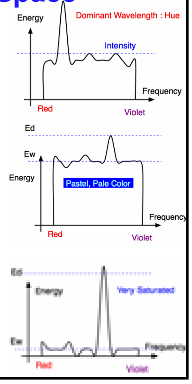
24

Review: HSV Color Space

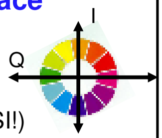
- hue: dominant wavelength, “color”
- saturation: how far from grey
- value/brightness: how far from black/white
- cannot convert to RGB with matrix alone







Review: YIQ Color Space



- color model used for color TV
  - Y is luminance (same as CIE)
  - I & Q are color (not same I as HSI!)
  - using Y backwards compatible for B/W TVs
  - conversion from RGB is linear

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.30 & 0.59 & 0.11 \\ 0.60 & -0.28 & -0.32 \\ 0.21 & -0.52 & 0.31 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

  - green is much lighter than red, and red lighter than blue

26

Review: Monitors

- monitors have nonlinear response to input
  - characterize by gamma
    - displayedIntensity = a^{γ} (maxIntensity)
- gamma correction
  - $\text{displayedIntensity} = \left(a^{1/\gamma}\right)^{\gamma}(\text{maxIntensity})$

    = a (maxIntensity)

27

Lighting I

28

Goal

model interaction of light with matter in a way that appears realistic and is fast

- phenomenological reflection models
  - ignore real physics, approximate the look
  - simple, non-physical
  - Phong, Blinn-Phong
- physically based reflection models
  - simulate physics
  - BRDFs: Bidirectional Reflection Distribution Functions

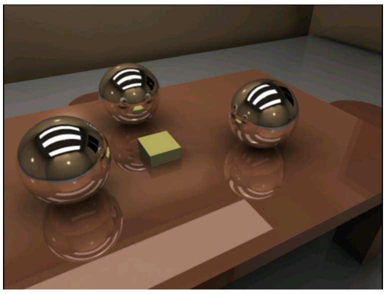
29

Photorealistic Illumination

[electricimage.com]

30

Photorealistic Illumination

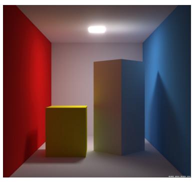


[electricimage.com]

Fast Local Illumination

Illumination

- transport of energy from light sources to surfaces & points
  - includes *direct* and *indirect illumination*



Images by Henrik Wann Jensen

Components of Illumination

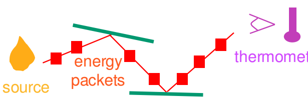
- two components: light sources and surface properties
- light sources (or *emitters*)
  - spectrum of emittance (i.e., color of the light)
  - geometric attributes
    - position
    - direction
    - shape
  - directional attenuation
  - polarization

Components of Illumination

- surface properties
  - reflectance spectrum (i.e., color of the surface)
  - subsurface reflectance
  - geometric attributes
    - position
    - orientation
    - micro-structure

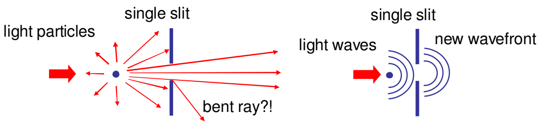
Illumination as Radiative Transfer

- radiative heat transfer approximation
  - substitute light for heat
  - light as packets of energy (photons)
    - particles not waves
  - model light transport as packet flow



Light Transport Assumptions

- geometrical optics (light is photons not waves)
  - no diffraction



  - no polarization (some sunglasses)
    - light of all orientations gets through
  - no interference (packets don't interact)
    - which visual effects does this preclude?

Light Transport Assumptions II

- color approximated by discrete wavelengths
  - quantized approx of dispersion (rainbows)
  - quantized approx of fluorescence (cycling vests)
- no propagation media (surfaces in vacuum)
  - no atmospheric scattering (fog, clouds)
    - some tricks to simulate explicitly
  - no refraction (mirages)
- light travels in straight line
  - no gravity lenses

Light Sources and Materials

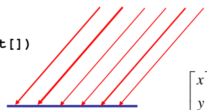
- appearance depends on
  - light sources, locations, properties
  - material (surface) properties
  - viewer position
- local illumination
  - compute at material, from light to viewer
- global illumination (later in course)
  - ray tracing: from viewer into scene
  - radiosity: between surface patches

Illumination in the Pipeline

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

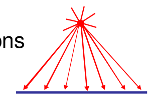
Light Sources

- types of light sources
  - `glLightfv(GL_LIGHT0,GL_POSITION,light[])`
- directional/parallel lights
  - real-life example: sun
  - infinitely far source: homogeneous coord w=0



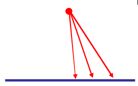
$$\begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix}$$

- point lights
  - same intensity in all directions



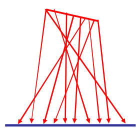
$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- spot lights
  - limited set of directions:
    - point+direction+cutoff angle



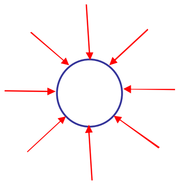
Light Sources

- 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)



Light Sources

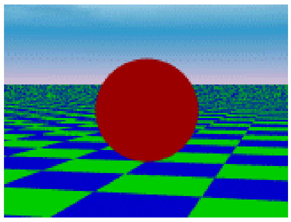
- ambient lights
  - no identifiable source or direction
  - hack for replacing true global illumination
    - (light bouncing off from other objects)



43

Ambient Light Sources

- scene lit only with an ambient light source



Light Position Not Important

Viewer Position Not Important

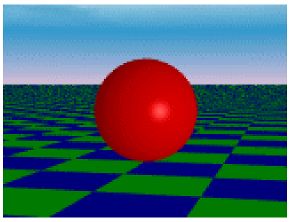
Surface Angle Not Important

44

Directional Light Sources

- scene lit with directional and ambient light

Surface Angle Important



Light Position Not Important

Viewer Position Not Important

45

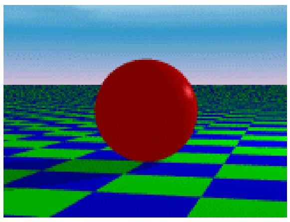
Point Light Sources

- scene lit with ambient and point light source

Light Position Important

Viewer Position Important

Surface Angle Important



46

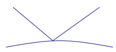
Light Sources

- 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

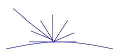
47

Types of Reflection

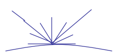
- *specular* (a.k.a. *mirror* or *regular*) reflection causes light to propagate without scattering.



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



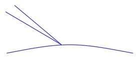
- *mixed* reflection is a weighted combination of specular and diffuse.



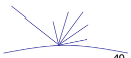
48

Types of Reflection

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



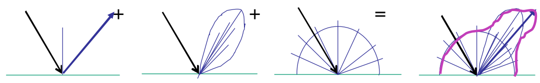
- *gloss* is the property of a material surface that involves mixed reflection and is responsible for the mirror like appearance of rough surfaces.



49

Reflectance Distribution Model

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



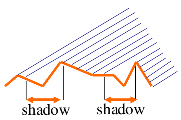
specular + glossy + diffuse =
reflectance distribution

50

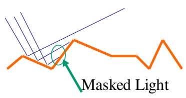
Surface Roughness

- at a microscopic scale, all real surfaces are rough

- cast shadows on themselves

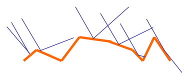


- "mask" reflected light:



51

Surface Roughness

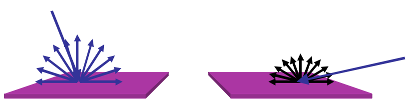


- notice another effect of roughness:
  - each "microfacet" is treated as a perfect mirror.
  - incident light reflected in different directions by different facets.
  - end result is mixed reflectance.
    - smoother surfaces are more specular or glossy.
    - random distribution of facet normals results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

52

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?



53

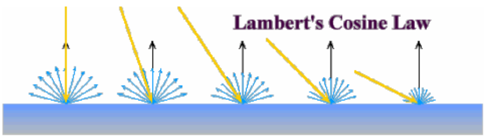
Lambert's Cosine Law

- ideal diffuse surface reflection

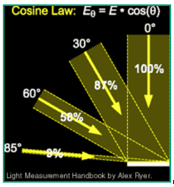
  the energy reflected by a small portion of a surface from a light source in a given direction is proportional to the cosine of the angle between that direction and the surface normal
- reflected intensity
  - independent of viewing direction
  - depends on surface orientation wrt light
- often called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

54

Lambert's Law

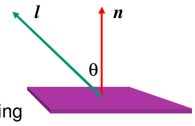


intuitively: cross-sectional area of the “beam” intersecting an element of surface area is smaller for greater angles with the normal.



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
  - $I_{diffuse} = k_d I_{light} (\mathbf{n} \bullet \mathbf{l})$



- 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

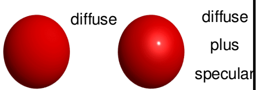
- Lambertian sphere from several lighting angles:

- 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

Lighting II

Specular Reflection

- shiny surfaces exhibit specular reflection
  - polished metal
  - glossy car finish



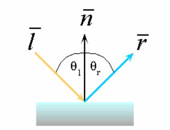
- specular highlight
  - bright spot from light shining on a specular surface
- view dependent
  - highlight position is function of the viewer's position

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

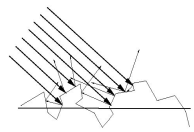


$\theta_{(l)ight} = \theta_{(r)eflection}$

61

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…



62

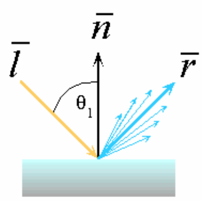
Empirical Approximation

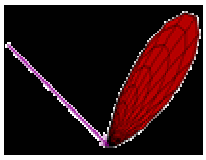
- we expect most reflected light to travel in direction predicted by Snell's Law
- but because of microscopic surface variations, some light may be reflected in a direction slightly off the ideal reflected ray
- as angle from ideal reflected ray increases, we expect less light to be reflected

63

Empirical Approximation

- angular falloff





- how might we model this falloff?

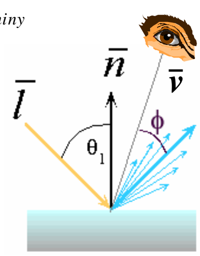
64

Phong Lighting

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

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

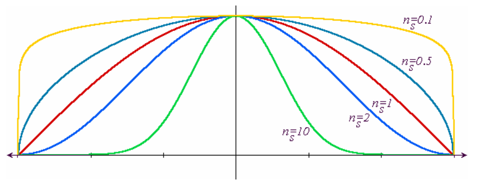
- n_{shiny} : purely empirical constant, varies rate of falloff
- k_s: specular coefficient, highlight color
- no physical basis, works ok in practice



65

Phong Lighting: The n_{shiny} Term

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



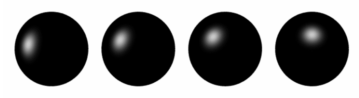
- *what does this term control, visually?*

Viewing angle – reflected angle

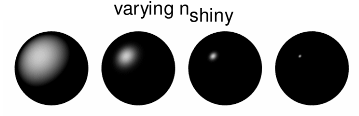
66

Phong Examples

varying I



varying n_{shiny}



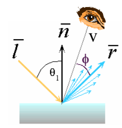
67

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

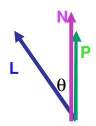


- how to efficiently calculate **r** ?

68

Calculating R Vector

P = **N** cos θ = projection of **L** onto **N**

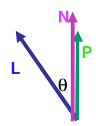


69

Calculating R Vector

P = **N** cos θ = projection of **L** onto **N**

P = N (N · L)



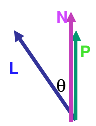
70

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)



71

Calculating R Vector

P = **N** cos θ |**L**| |**N**| projection of **L** onto **N**

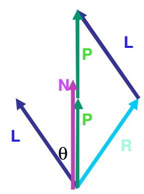
P = **N** cos θ **L, N** are unit length

P = N (N · L)

2 P = R + L

2 P – L = R

2 (N (N · L)) - L = R



72

Phong Lighting Model

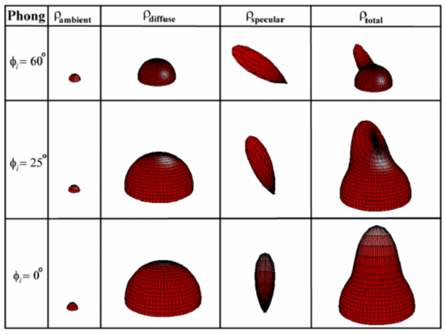
- combine ambient, diffuse, specular components

$$\mathbf{I}_{\text{total}} = \mathbf{k}_s \mathbf{I}_{\text{ambient}} + \sum_{i=1}^{\#lights} \mathbf{I}_i (\mathbf{k}_d (\mathbf{n} \bullet \mathbf{l}_i) + \mathbf{k}_s (\mathbf{v} \bullet \mathbf{r}_i)^{n_{shiny}})$$

- commonly called *Phong lighting*
  - once per light
  - once per color component

73

Phong Lighting: Intensity Plots



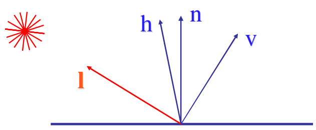
74

Blinn-Phong Model

- variation with better physical interpretation
  - Jim Blinn, 1977

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

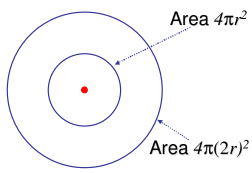
- ***h***: halfway vector
  - h must also be explicitly normalized: h / |h|
  - highlight occurs when h near n



75

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



76

Light Source Falloff

- non-quadratic falloff
  - many systems allow for other falloffs
  - allows for faking effect of area light sources
  - OpenGL / graphics hardware
    - I_0: intensity of light source
    - ***x***: object point
    - r: distance of light from ***x***

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

77

Lighting Review

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

78

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)

79

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/

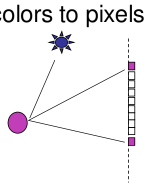
80

Shading

81

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?)



82

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

83

Applying Illumination

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

84

Flat Shading

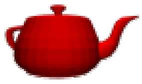
- simplest approach calculates illumination at a single point for each polygon

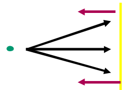
- obviously inaccurate for smooth surfaces

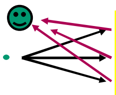
85

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





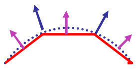


86

Improving Flat Shading

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

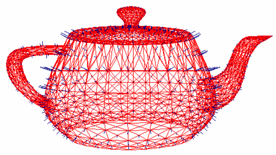
- for smoother-looking surfaces we introduce *vertex normals* at each vertex
  - usually different from facet normal
  - used *only* for shading
  - think of as a better approximation of the *real* surface that the polygons approximate



87

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

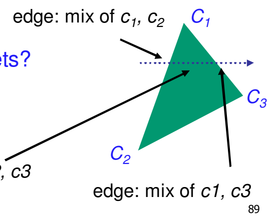


88

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

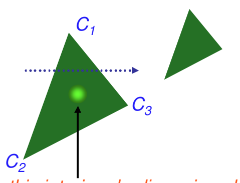
does this eliminate the facets?

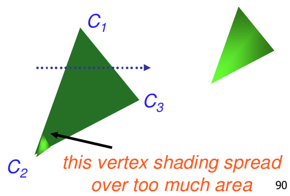


89

Gouraud Shading Artifacts

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

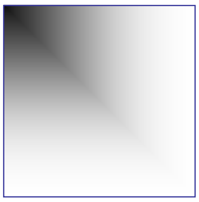




90

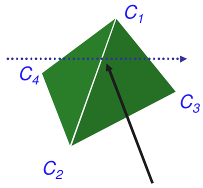
Gouraud Shading Artifacts

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

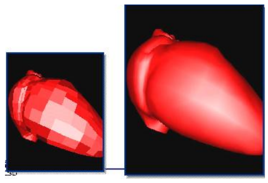


Gouraud Shading Artifacts

- Mach bands

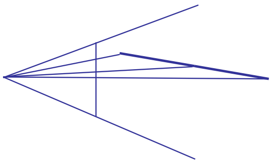


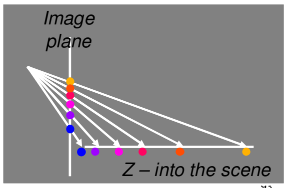
Discontinuity in rate of color change occurs here



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

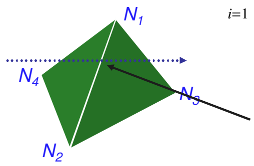
- **not** the same as Phong lighting
  - common confusion
  - Phong lighting: empirical model to calculate illumination at a point on a surface

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

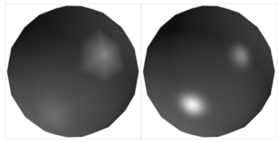
Phong Shading Difficulties

- 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 hardware
  - but can be simulated with texture mapping

97

Shading Artifacts: Silhouettes

- polygonal silhouettes remain

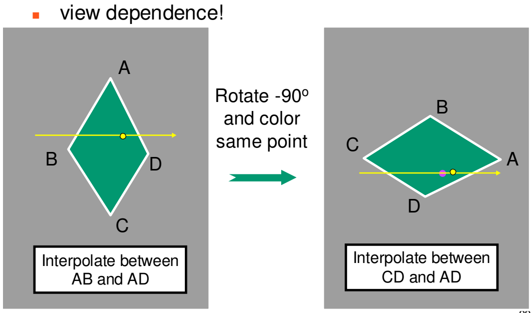


Gouraud *Phong*

98

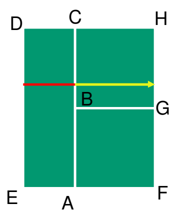
Shading Artifacts: Orientation

- interpolation dependent on polygon orientation
  - view dependence!



99

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

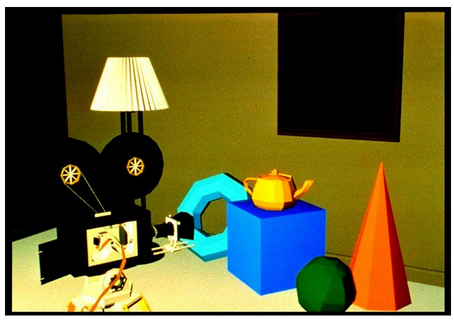
100

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

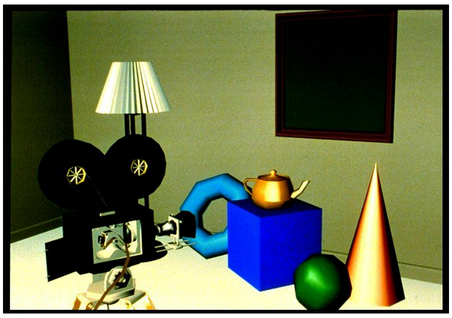
101

Shutterbug: Flat Shading

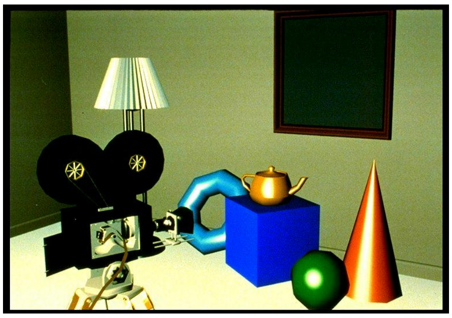


102

Shutterbug: Gouraud Shading

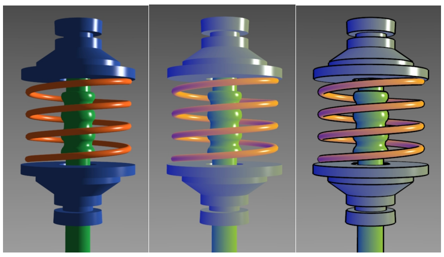


Shutterbug: Phong Shading



Non-Photorealistic Shading

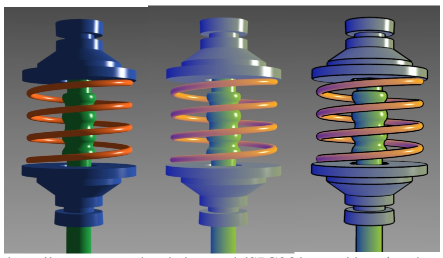
- cool-to-warm shading $k_w = \frac{1+\mathbf{n}\cdot\mathbf{l}}{2}, c = k_w c_w + (1-k_w)c_c$



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

Non-Photorealistic Shading

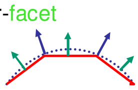
- draw silhouettes: if $(\mathbf{e}\cdot\mathbf{n}_0)(\mathbf{e}\cdot\mathbf{n}_1) \le 0$, **e**=edge-eye vector
- draw creases: if $(\mathbf{n}_0\cdot\mathbf{n}_1) \le \textit{threshold}$

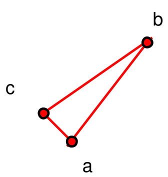


http://www.cs.utah.edu/~gooch/SIG98/paper/drawing.html

Computing Normals

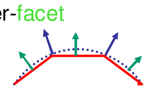
- per-vertex normals by interpolating per-facet normals
  - OpenGL supports both
- computing normal for a polygon

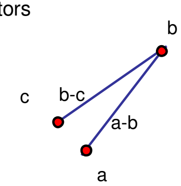




Computing Normals

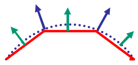
- per-vertex normals by interpolating per-facet normals
  - OpenGL supports both
- computing normal for a polygon
  - three points form two vectors

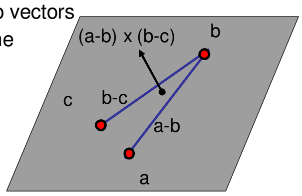




Computing Normals

- 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

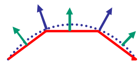


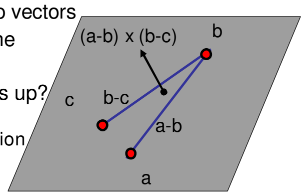


109

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
  - which side of plane is up?
    - counterclockwise point order convention





110

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);
```

111