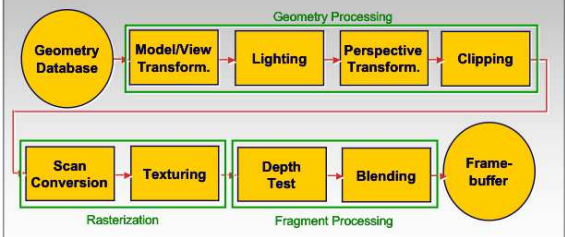
Lighting & Shading

CPSC 314

The Rendering Pipeline



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 Rendering 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

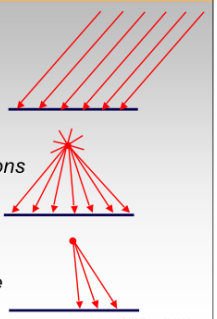
Materials

- Simple, non-physical reflection models

Light Sources

Types of light sources

- Directional/parallel lights
  - *E.g.sun*
  - *Homogeneous vector*
- (Homogeneous) point lights
  - *Same intensity in all directions*
  - *Homogeneous point*
- Spot lights
  - *Limited set of directions*
  - *Point+direction+cutoff angle*



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.

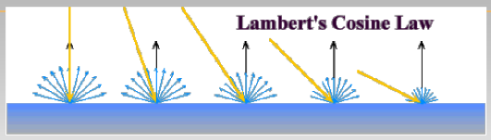
Reflectance Distribution Model

Most surfaces exhibit complex reflectances

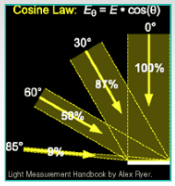
- Vary with incident and reflected directions.
- Model with combination

specular + glossy + diffuse = reflectance distribution

Lambert's "Law"

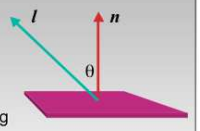


Intuitively: cross-sectional area of the "beam" intersecting an element of surface area is smaller for greater angles with the normal.



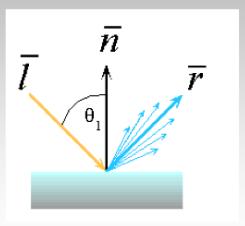
Computing Diffuse Reflection

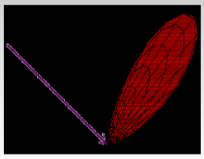
- Depends on angle of incidence: angle between surface normal and incoming light
  - $I_{diffuse} = k_d I_{light} \cos \theta$
- In practice use vector arithmetic
  - $I_{diffuse} = k_d I_{light} (n \bullet l)$
- Always normalize vectors used in lighting
  - *n, l should be unit vectors*
- Scalar (B/W 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)*



Empirical Approximation

Angular falloff





how might we model this falloff?

Phong Lighting

Most common lighting model in computer graphics

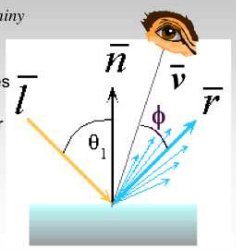
- *(Phong Bui-Tuong, 1975)*

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

n_{shiny}: purely empirical constant, varies rate of falloff

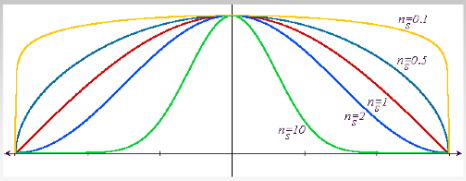
k_s: specular coefficient, highlight color

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



what does this term control, visually?

Viewing angle – reflected angle

Phong Examples

varying l

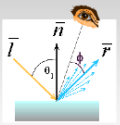
varying n_{shiny}

Calculating Phong Lighting

compute cosine term of Phong lighting with vectors

$$\mathbf{I}_{\text{specular}} = \mathbf{k}_s \mathbf{I}_{\text{light}} (\mathbf{v} \bullet \mathbf{r})^{n_{shiny}}$$

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



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 );
```


Shading

CPSC 314

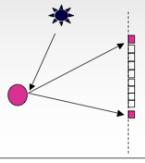
Lighting vs. Shading

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 computing pixel colors



Applying Illumination

Lighting:

- 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

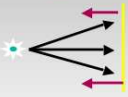
If an object really is faceted, is this accurate?

Flat Shading Approximations

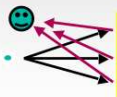
If an object really is faceted, is this accurate?

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

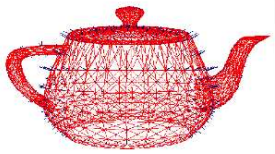
For smoother-looking surfaces we introduce vertex normals at each vertex

- 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

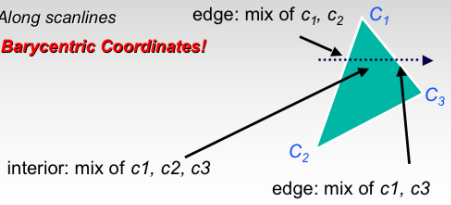


Gouraud Shading

Most common approach, and what OpenGL does

- Perform Phong lighting at the vertices
- Linearly interpolate the resulting colors over faces
  - *Along edges*
  - *Along scanlines*

Same as Barycentric Coordinates!



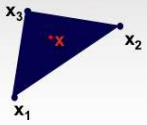
Barycentric Coordinates

- Convex combination of 3 points

$$\mathbf{x} = \alpha \cdot \mathbf{x}_1 + \beta \cdot \mathbf{x}_2 + \gamma \cdot \mathbf{x}_3$$

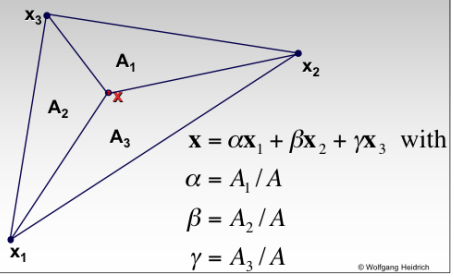
$$\text{with } \alpha + \beta + \gamma = 1,\ 0 \le \alpha, \beta, \gamma \le 1$$

- α, β, and γ are called *barycentric coordinates*



Barycentric Coordinates

One way to compute them:



$$\mathbf{x} = \alpha\mathbf{x}_1 + \beta\mathbf{x}_2 + \gamma\mathbf{x}_3 \quad \text{with}$$

$$\alpha = A_1 / A$$

$$\beta = A_2 / A$$

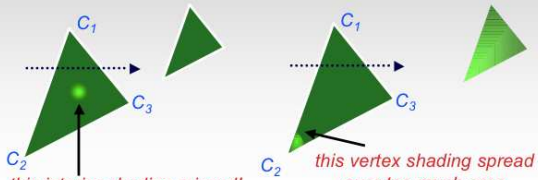
$$\gamma = A_3 / A$$

Gouraud Shading Artifacts

often appears dull, chalky

lacks accurate specular component

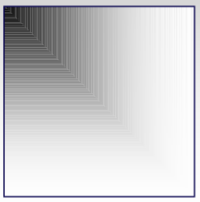
- if included, will be averaged over entire polygon



Gouraud Shading Artifacts

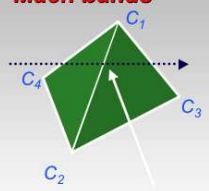
Mach bands

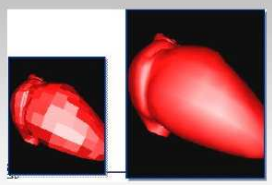
- Eye enhances discontinuity in first derivative
- Very disturbing, especially for highlights



Gouraud Shading Artifacts

Mach bands



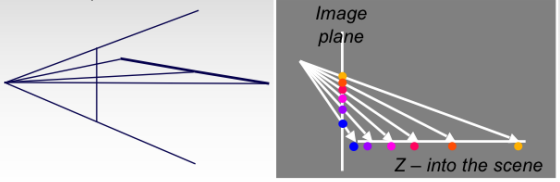


Discontinuity in rate of color change occurs here

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

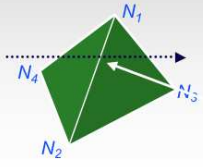
Phong Shading

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

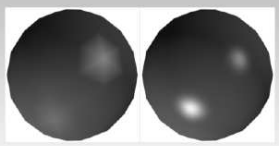
- Messes up the angles between vectors
- Have to keep eye-space vectors around

no direct support in hardware

- But can be simulated with texture mapping

Shading Artifacts: Silhouettes

Polygonal silhouettes remain

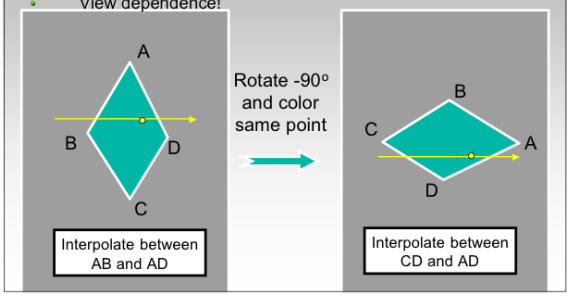


Gouraud *Phong*

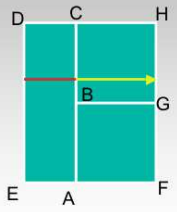
Shading Artifacts: Orientation

Interolation 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

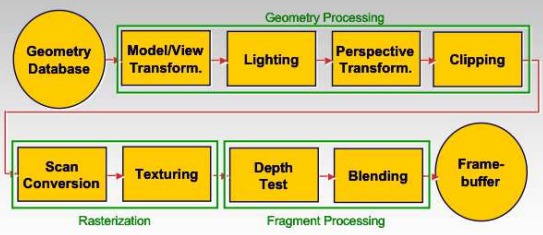
Gouraud shading

- 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

The Rendering Pipeline



Coming Up

Tuesday:

- Shading (2)
- Quiz 1

Thursday:

- Clipping
- A1 due